Budapesti Műszaki és Gazdaságtudományi Egyetem
Közlekedésmérnöki Kar
Közlekedésautomatikai Tanszék

IP alapú irányítórendszer modellezése és fejlesztése ACTROS típusú forgalomirányító berendezésekhez

Szakdolgozat

.....
Orsó Tamás
EHAAWJ

2010.

Szakdolgozatomban egy, az általam tervezett és fejlesztett közúti közlekedési irányítórendszer kerül bemutatásra. Ennek előkészítéseként, általánosságban leírásra kerültek a forgalomirányító rendszerek működéseinek céljai, felépítéseinek módjai, valamint az elemeinek, vagyis a központoknak és a csomóponti berendezéseknek feladatai, és a közöttük lévő kapcsolat kiépítésének lehetőségei is. A fejlesztési tendenciák vizsgálatával és figyelembevételével került modellezésre és kidolgozásra az új rendszer. A létrehozott rendszer struktúráját tekintve, centralizált, így a forgalomirányításra vonatkozó döntések a központban születnek, amik az adott csomópontra vonatkozó, detektoradatokon, vagyis a mért járműszámokon, valamint a korlátozó feltételként megadott adatokon alapulnak. A rendszer másik jellemző tulajdonsága, hogy a terepi berendezés, ami jelen esetben egy ACTROS VTC-3000 készülék, és a központ között IP alapú kapcsolaton keresztül történik az adatcsere.

Kulcsszavak*: központi, adaptív, IP alapú, forgalomirányítás*

Konzulens: **Tettamanti Tamás, egyetemi tanársegéd**

Tartalomjegyzék

1. Bevezetés

Napjaink közúti közlekedésére jellemző, a pályát igénybe vevők számának folyamatos növekedése és az ezzel járó hatások, mint például a torlódások, a balesetveszély, a környezetterhelés fokozódása is. Az egyre szűkösebb útkapacitások problémájára, megoldást jelenthet a pálya bővítésére irányuló intézkedések. Azonban gazdaságosabb és környezetkímélőbb, ha ehelyett igénybe vesszük a korszerű forgalomirányítás eszközeit és elgondolásait, melyek segítségével, az aktuális forgalmi állapotoknak megfelelő beavatkozásokat tehetünk.

A forgalomfüggő irányításhoz szükség van az aktuális közlekedési helyzet ismeretére, melyeket a pályára telepített mérőberendezések biztosítanak. A szerzett adatok alapján a döntés megszületése történhet helyileg, vagy egy forgalomirányító központban is, valamilyen forgalomtechnikai cél megvalósítása érdekében.

A jelenlegi fejlesztési tendenciák, egy olyan lehetséges jövőképet mutatnak, ahol a központok nagyobb szerepet kapnak az irányításban, tehát a feladatok nagy részének elvégzése ezeket fogja terhelni. Emellett, a csomóponti eszközökkel történő kommunikációban, egyre inkább általánosabb, elsősorban IP alapú megoldások lesznek elterjedtebbek.

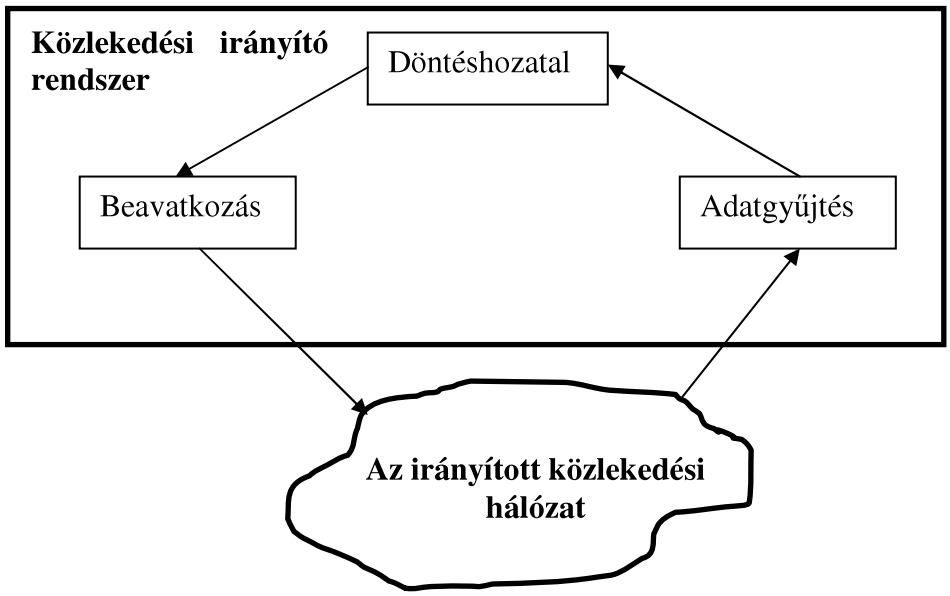
Szakdolgozatom célja egy, az ehhez hasonló rendszer működésének felvázolása és kivitelezésének bemutatása lesz. A rendszer megalkotásakor a cél az volt, hogy olyan igényeket is kielégítsen, amelyeket ilyen rendszerektől, minimális szinten el lehet várni:

A központi szoftver a kapott járműadatokra, kompletten kidolgozott fázistervvel válaszol. A számítások során a hálózat egészét figyelembe véve, képes a ciklusidők módosítására is, az optimális forgalmi szint elérése érdekében. Emellett a forgalomtechnikai adatok archiválásra is fel van készítve, a későbbi ellenőrzések és lekérdezések érdekében. A kezelő személy számára egyszerű és könnyen kezelhető felületet biztosít, az irányítási folyamatokba való beavatkozáshoz, valamint a lementett adatok előhívásához.

2. Forgalomirányító rendszerek

A közúti forgalomirányító rendszerek a forgalom biztonságos, gazdaságos és környezetkímélő módon történő lebonyolításáért felelnek. A forgalmi állapotok alapján hozzák meg a döntéseket, melyek által tudják a folyamatot szabályozni, így elmondható, hogy a forgalomirányítás zárt szabályozó körben történik (1. ábra).

A megfelelő szabályozáshoz, tartalmazniuk kell valamilyen visszacsatolást, mely által az aktuális állapot így az előző beavatkozás hatásait tudják érzékelni. Ez az elv, függetlenül a szabályozott terület méretétől, egyaránt érvényes a forgalomirányító központokra, illetve a helyi berendezésekre is; az eltérés az alkalmazott célfüggvényeknél van [1].



1. ábra: Általános közlekedési szabályozás
forrás: [1]

2.1. Célkitűzések

- Gazdaságosság növelése, költségek csökkentése. „Az idő pénz”, így fontos, hogy minél kevesebb időt töltsenek a járművek a hálózaton, illetve, fontos, hogy csökkentsük a nem helyváltoztatással töltött idők számát, vagyis a várakozási időket. Ezáltal kevesebb üzemanyag kerül feleslegesen felhasználásra, az utazási költségek is csökkenhetnek.
- Forgalombiztonság növelése. Napjaink közlekedéspolitikájában, hangsúlyos szerepe van a balesetek számának alacsonyabb szinten tartásának, illetve azok súlyosságának enyhítésének. Ez különösen fontos, hiszen világtendencia az egyre növekvő helyváltoztatási igény.
- A környezetvédelem fontossága. Jelentősége a növekvő mobilitással együtt növekszik, a globális (pl.: éghajlatváltozás) és lokális (pl.: egészségügyi problé-

mák) hatásoknak köszönhetően. A környezetterhelés csökkentésében is szerepet kell vállalni: optimalizált működésükkel az irányító rendszereknek hozzájárulnak, a felesleges energiafelhasználás, káros anyag kibocsátás és zajterhelés mérsékléséhez. Ez többek között az utazási és várakozási idők csökkentésével érhetik el.

A leírt főszempontok között van kapcsolat, amit mi sem bizonyít jobban, a következőkben felsorolt, ezek megvalósítását szolgáló intézkedések. Nem lehet egyértelműen egy bizonyos szemponthoz kötni azokat, mindegyikkel vonható párhuzam:

- Közlekedéssel kapcsolatos információszolgáltatás javítása. Kellő információellátással lerövidülnek az út és parkolóhely keresések.
- Adott útszakaszok kapacitásainak maximális kihasználása, vagy éppen az ellentettje: a tehermentesítés.
- Közösségi közlekedés előnyben részesítése: különböző intézkedésekkel, beavatkozásokkal ösztönözni kell az embereket a közösségi közlekedési eszközök használatára, az egyéni gépjárművekkel szemben.
- Forgalmi zavarok megelőzése, gyors megszűntetése.

Az irányítórendszerek működésük során különböző forgalomtechnikai paraméterek optimális értéken tartására törekszenek. A prioritás esetenként más és más lehet, ezt a közlekedéspolitika határozza meg. Az optimalizálandó paraméterek a következők lehetnek:

- Átbocsátó képesség maximalizálása
- Utazási sebesség maximalizálása, vagy utazási idő minimalizálása
- Megállások, illetve gyorsítások és lassítások számának minimalizálása
- Sorhosszak minimalizálása
- Energiafelhasználás minimalizálása
- Környezetterhelés, tehát káros anyag és zaj szennyezés minimalizálása

A közúti forgalomirányító rendszer hatékonyságát, amely függ az alkalmazott célfüggvénytől, különböző mérőszámokkal tudjuk jellemezni:

- Teljes utazási idő (Total Time Spent - TTS), vagyis a hálózaton eltöltött összes idő. Mértékegysége: jármű·időegység, általában jármű·óra.
- Teljes utazási távolság (Total Travel Distance - TTD), vagyis a hálózaton megtett teljes távolság. Mértékegysége: jármű·hosszegység, általában jármű·km.
- A teljes utazási távolság és a teljes utazási idő hányadosa megadja az átlagos utazási sebességet (Mean Speed - MS). Mértékegysége: hosszegység/időegység, általában km/h.

2.2. Az irányító rendszer felépítése

A rendszer irányítása és felügyelete a központban történik, ehhez azonban megfelelő információkkal (detektoradatok, baleseti jelentések, meteorológiai információk) kell ellátni, melyek által megszülethet a beavatkozáshoz szükséges döntés. A forgalomirányító központok, berendezések és az adatgyűjtő rendszer egészét forgalomirányító rendszernek nevezzük.

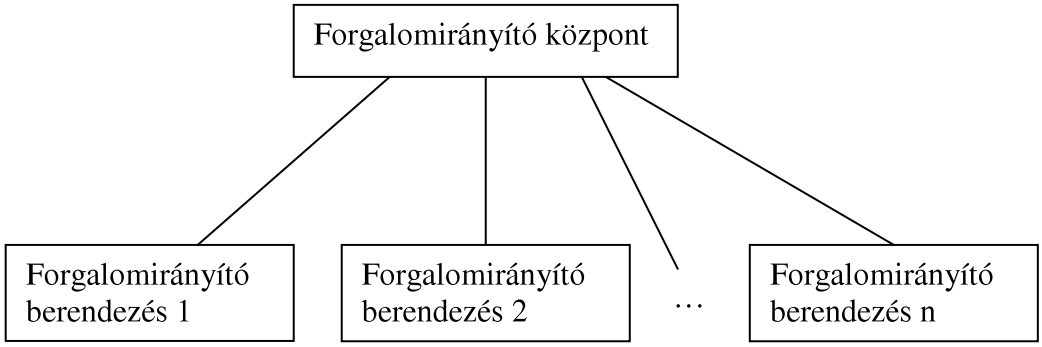
Az irányító rendszernek alapvetően háromfajta struktúráját különböztetjük meg:

- Centralizált, teljesen hierarchikus
- Decentralizált, teljesen elosztott
- Vegyes elrendezésű, vagyis az előző két struktúra keveréke

Centralizált

Ennél a struktúránál, az egyes forgalomirányító berendezések működését, a helyi csomópont mérőeszközeitől származó adatokra épülő döntések alapján, a központ szabályozza (2. ábra). A központi számítógép, a döntést a rajta futó optimalizáló algoritmus és meghatározott paraméterek alapján végzi.

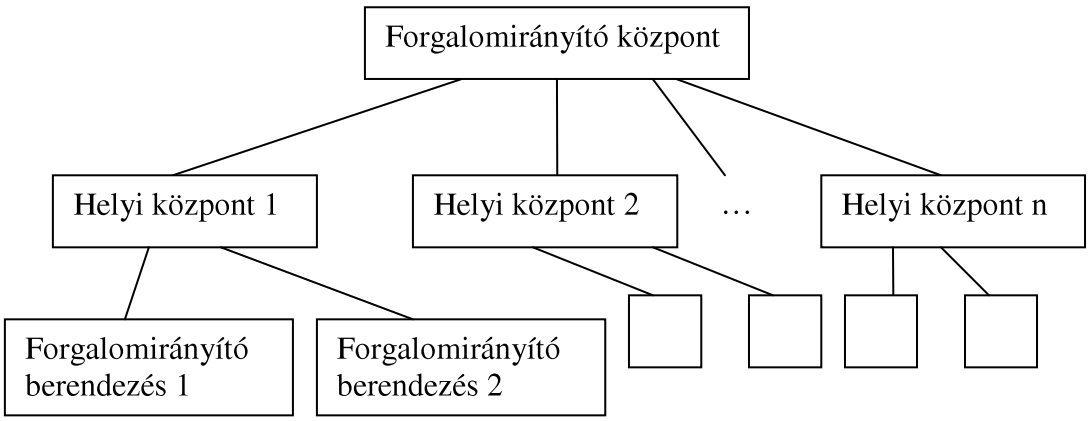
Arra az esetre, ha a központ inaktív állapotba kerül, vagy olyan hálózati probléma lép fel, amely során a központ és a berendezés közötti kapcsolat megszakad, a csomóponti berendezéseknek egy szükségprogramot kell tartalmazniuk, hogy a minimális forgalomszabályozást fenn lehessen tartani.



2. ábra: Centrális forgalomirányító rendszer

Decentralizált

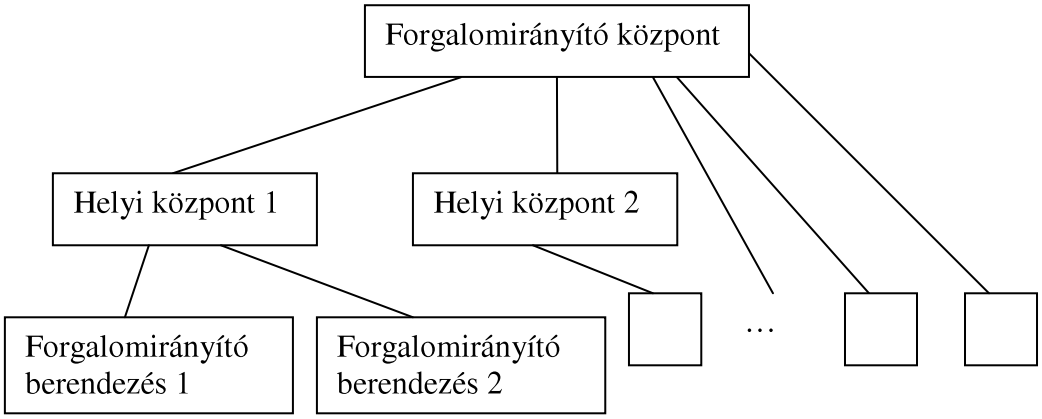
Az elosztott rendszereknél a döntés meghozása már nem a központban történik, hanem helyileg, a forgalomirányító berendezésekben. Ezáltal a központnak nem kell a komoly számításokat végezni, feladata a teljes rendszer összefogására, menedzselésére és a helyi készülékek stratégiai információkkal ellátására korlátozódik (3. ábra).



3. ábra: Teljesen elosztott rendszer

Vegyes elrendezésű

Gyakorlatilag, nem léteznek csak centralizált és csak decentralizált rendszerek, általában a kettő sajátos keveréke van használatban. A csomóponti berendezések területi alközpontokhoz vagy csoportvezérlőkhöz csatlakoznak, amik viszont egy főközponttal vannak kapcsolatban. (4. ábra).



4. ábra: A vegyes elrendezésű rendszer

A Budapesti Forgalomirányító Központ is vegyes felépítésű rendszer, a központi egység mellett négy alközpont vesz rész az irányításban:

- Centrum(Szabó Ervint tér)
- Észak Pest (Váci út – Róbert Károly körút kereszteződés)
- Dél Pest (Határ úti gyalogos aluljáró)
- Dél Buda (Szerémi út aluljáró)

2.3. A forgalomirányító központok

Több forgalomirányító berendezés munkáját célszerű egymással összehangolni az adott terület, vagy akár a teljes hálózat optimális működése érdekében. Az összehangolási feladat egy központban történik, meghatározott célok elérése érdekében. Az irányítás során fontos a célkitűzés megállapítása, vagyis az optimalizálás milyen feltételek mellett menjen végbe. Ezek közül néhány lehetséges szempont:

- A meglévő útkapacitások kihasználása
- Egyenletes forgalomlefolyás biztosítása
- Kibocsátás csökkentés
- Közösségi közlekedési eszközök előnyben részesítése

2.3.1. A forgalomirányító központokkal szemben támasztott követelmények

A forgalomirányító központoknak különböző szempontoknak kell megfelelniük:

Forgalomtechnikai

A különböző forgalomtechnikai-jellemzők javítására különböző intézkedéseket lehet tenni:

- az utazási idők csökkentése
- az úthálózaton eltöltött idők csökkentése
- megállások, lassítások-gyorsítások számának csökkentése
- várakozási idők csökkentése
- átbocsátott forgalom maximalizálása
- utazási, illetve szállítási költségek csökkentése
- környezetterhelés csökkentése

Biztonság, megbízhatóság

A közlekedéspolitika egyik fontos pontja közúti közlekedéssel kapcsolatos balesetek számának és azok súlyosságának csökkentése, ez csak úgy érhető el, ha

- a rendszer rendelkezésre állása a lehető legnagyobb, vagyis üzemen kívül eltöltött idők, vagyis a karbantartási és javítási idők minimálisak legyenek
- többszintű hibavédelem megléte szükséges
- a rendszer biztonsági szintje nem alacsonyabb a forgalomirányítástól elvárt szinttől

Rugalmasság

Fontos jellemző az alkalmazkodó képesség, a mi meghatározza, hogy a rendszer tud-e alkalmazkodni egy fennálló állapothoz, és milyen gyorsan.

- Jelzéstervek, jelzési idők változtathatósága
- A hálózat szervezésének változtathatósága
- Irányítási stratégia változtathatósága
- A változó igényekhez való igazodás
- A váratlan események bekövetkezésénél történő beavatkozási lehetőség

Gazdasági szempontok

Lényeges tényező, hogy a rendszernek milyen költségvonzata van a teljes életciklusát tekintve.

- Tervezési költségek
- Kiépítési költségek
- Üzemeltetési költségek
- Karbantartási költségek
- Javítási költségek
- Egyéb, járulékos költségek

Egyéb

- Műszaki megvalósíthatósági elvárások
- Környezetvédelmi szempontok

Természetesen a fenti szempontok mindegyikének eleget tenni, a prioritások esetenként mások lesznek. Az egyes követelmények fontosságának jellemzésére mérőszámokat használhatunk, ezt multikritériumos irányításnak nevezzük. Az egyes (K_i) kritériumok fontosságát (S_i) súlyozó-tényezőkkel állapítjuk meg. A súly-tényezők változnak, értéküket a közlekedéspolitika is befolyásolja.

$\sum_{i=1}^{n} S_i K_i = \sum J \rightarrow \min$ Ahol $\sum_{i=1}^{n} S_i = 1$ Vagyis a súlyozó-tényezők együtt 100 %-ot adnak.

Alkalmazási helytől és igényektől függően, a lehető legoptimálisabb variáció alkalmazására kell törekedni, amivel a rendszerre vonatkozó ráfordítások minimalizálhatók.

2.3.2. Forgalomirányítási stratégiák

Négy-fajta forgalomirányítási stratégia alakult ki az évek során, ezek a következők megjelenésük időrendjében [2] [3]:

- Kézi vezérlésű
- Időterv vezérlésű
- Számítógépes programválasztó (off-line)
- Számítógépes programalkotó (on-line)

2.3.2.1. Kézi vezérlésű

A központba (kezelő terembe) bevezetett információk alapján, a kezelőszemélyzet dönt a programváltásokról. A forgalomról, az időjárásról, a berendezések állapotáról szóló információk egy arra alkalmas tablón, (zárt láncú TV) jelennek meg.

Az állandó szakszemélyzetnek köszönhetően állandó felügyelet van, ezért váratlan események bekövetkezésénél gyors beavatkozásra nyílik mód. A döntések aktuális adatok alapján történnek, ezért kimondhatjuk, hogy a rendszer forgalomfüggő.

Ennek a stratégiának alkalmazása, tekintve az állandó személyzetet, a költséges berendezéseket, nem gazdaságos. A rendszer alacsony színvonalú, áttekinthetetlen és mára már elavult.

2.3.2.2. Időterv szerinti vezérlés

Egyel fejlettebb irányítási színvonalat képviselnek az időterv szerinti stratégiát alkalmazó központok. Ennek egy automatikus kapcsolóóra az alapja, amely meghatározott időközönként elvégzi a programváltásokat. Ezáltal az állandó szakszemélyzet, illetve a TV-hálózat vesztett a jelentőségéből, már nem szükséges a jelenlétük, elhagyásukkal a költségek csökkenhetnek. Ez, azonban ahhoz vezet, hogy a rendszer kevésbé lesz rugalmas, képtelen lesz a váratlan helyzetekre megfelelően reagálni.

A rendszer működésének elengedhetetlen feltétele a pontosan kidolgozott időterv, amit rendszeresen kell aktualizálni, bár ennek ellenére sem tükrözi mindig a valóságot, ráadásul az időterv folyamatos frissítése jelentős költségvonzattal is jár.

2.3.2.3. Programválasztás

Az ilyen stratégiát alkalmazó központokat off-line központoknak is nevezik. A mérőberendezések felől kapott forgalmi adatok alapján a központban, az adott csomópontra vonatkozóan, egy adatbázisból történnek az aktuális forgalmi helyzetnek leginkább

megfelelő programterv kiválasztásai és kivezérlései a csomópontokhoz. Ez az adatbázis több eshetőségre is tartalmaz programtervet, még a ritkán bekövetkezendő helyzetekre (pl.: balesetek, útvonal-lezárások, terelések, rendkívüli időjárás következtében kialakult zavarok,) is léteznek speciális, átmeneti tervek. Ezek alapján, a rendszerről elmondható, hogy forgalomfüggő, de hátránya, hogy nem feltétlenül van minden esetre olyan programterv, ami a valóságnak maximálisan megfelel.

A rendszer fontos része a megfelelő helyzetfelismerés, melynek működését szigorúan kell ellenőrizni, különböző biztonsági szoftverek segítségével, a hibás döntések elkerülése végett.

2.3.2.4. Programalkotás

A számítógépes programalkotó központokat, - amiket on-line központoknak is neveznek - képviselik a forgalomirányítás legmagasabb szintjét. A központok, az aktuális forgalmi adatoknak megfelelően, képesek az érvényben lévő programon változtatni.

A programalkotó stratégiának két nagyobb típusa van:

- Fázis távvezérlésű
- Jelzőcsoport távvezérlésű

Fázis távvezérlésű

A központ az adott berendezés fázistervei közül válogatja ki az adott forgalmi helyzethez leginkább megfelelő programot, melynek életbe léptetése a programban kijelölt STOP pontokon keresztül történik. Adott programstruktúrán belül mód nyílik a periódusidő és fázisok kezdetének és hosszának módosítására

Jelzőcsoport vezérlésű

A jelzőcsoport vezérlésű irányelv képviseli a legrugalmasabb, a leginkább az aktuális forgalomhoz igazodó stratégiát. Az egyes berendezések tervei a központban találhatóak, melyeket a mérőberendezés adatai alapján tud módosítani a forgalmi helyzetnek megfelelően, így terepi beavatkozásra nincs szükség.

A központ önmagában még nem elég, hatékonysága függ a rajta futó szoftverektől és az azoknak megadott irányítási paraméterek helyességétől. Ennek érdekében a központi számítógépen több ellenőrző és hihetőség-vizsgáló alprogram fut. A rendszer legnagyobb előnye, a forgalomfüggőség mellett, a rugalmassága; viszont az alkalmazásának komoly technikai és anyagi igényei vannak.

2.4. Forgalomirányító berendezések

A csomópontokon áthaladó forgalom biztonságos lebonyolításában nagy szerepük van a forgalomirányító berendezéseknek. Ezek a terepi eszközök állapítják meg az adott csomópont jelzőcsoportjainak jelzésképeit, melyeket vagy önállóan, a beépített logika segítségével állítanak elő, vagy a központ felől kapják meg. A fennálló forgalmi helyzethez való viszonyukat tekintve, lehetnek fixprogramosak és forgalomfüggők. Az utóbbi eset mérőeszközök (detektorok) meglétét feltételezi.

Néhány elterjedtebb forgalomirányító berendezés:

- Nikola Tesla [4]
- FB016 [5]
- C800, C900; [6]
- VSF-6, -12, -24, -36; -48 [7]
- SGS 32/40, [8]
- VTC 3000, [9]

Ezek terepi készülékek közül bővebben, az ACTROS VTC-3000 készülékről lesz szó, mivel a jelen dolgozat keretében készített programok egy része, ennek a berendezésnek, mint kliensnek, készültek.

2.4.1. ACTROS

A Signalbau Huber által készített ACTROS VTC 3000, egy korszerű forgalomirányító berendezés, mely igényes és intelligens forgalomtechnikai szabályozást biztosít. Tervezésénél figyelembe vették a jelen és a várható, jövőbeli közlekedésre vonatkozó követelményeket is. Felépítése moduláris, ami fokozza a rendszer rugalmasságát, és ezáltal könnyebb a megrendelők egyedi igényeihez alkalmazkodni [10].

Felhasználóbarát és hatékony megvalósítás:

- Gyártó-specifikus programnyelv helyett, általánosabb célú JAVA használata a készülék programozása során
- Internet-technológia alkalmazása
- OCIT-támogatás
- Felhasználói adatok és programok viszonylag egyszerű fel és letöltése
- Diagnosztikus lehetőségek
- Hibatároló funkció

A berendezés rendszer-hardverei

- CPU LS2000 vezérlőegység. Elemei: különböző típusú memóriák, órajeladó és különböző interfészek (szerviz, ethernet, egyéb)
- DCF rádióóra modul: rendszeridő előállítására szolgál. Az időadatokat rádión keresztül kapja.
- CPU 020. Elemei: különböző típusú memóriák, processzor, óra és képernyőinterfész.
- CPU-V55 ellenőrző egység: a lámpaadatokat ellenőrzi, összevetve a referenciaadatokkal.
- NKK hálózati ellenőrző kártya: feladata többek között, a hálózati és rendszerfeszültség mérése és ellenőrzése
- ASK automata és védelmi kártya. A hálózatra való fel és lekapcsolódást irányítja

A berendezés funkció-hardverei

- SK 24 kapcsolókártya: feladata a jelzőberendezések jelkimeneteinek ellenőrzése és vezérlése. Ez a kártya 24 szabadon konfigurálható jelcsatorna irányítására alkalmas
- IO 24 ki-és bemeneti kártya: Ez egy kombinált ki- és bemeneti kártya. A létrehozott csatornák (bemeneti) legtöbbször detektorokhoz vannak rendelve.

3. Kommunikáció a központ és a helyi berendezés között

3.1. Ipari kommunikációs hálózatok hierarchiája

Napjainkban, az ipari hálózati kommunikációk hierarchikusan épülnek fel. Általában három hierarchia szintet különböztetünk meg [25]:

- Eszköz szint (detektorok, bejelentkezők, jelzőlámpák)
- Üzemi szint (forgalomirányító berendezés)
- Igazgatási vagy irodai szint (forgalomirányító központ, távfelügyeleti rendszerek)

Az eszköz szint az elosztott terepi berendezések, a mérőberendezések és a beavatkozó eszközök szintje. Az egyes egységek hálózatba sorolására speciális, illetve általános célú terepi buszokat alkalmaznak. Az elemek felügyeletéért intelligens vezérlő egységek (PLC-k) felelnek, melyek már a következő szinten, vagyis az üzemi szinten helyezkednek el. A vezérlők közti kommunikációért, kisebb rendszerekben terepi buszok, nagyobb rendszerben pedig IP alapú megoldások felelnek. Ezen a szinten történnek az eszközök felől érkező adatok összefogása és gyűjtése, valamint a rendszer működését befolyásoló beavatkozások is. A legfelső szint az igazgatási szint, itt születnek a stratégiai döntések, valamint itt történnek az adatok archiválása és az egyéb adatszolgáltatások is. Ez a szint kizárólag Ethernet alapú (TCP/IP hálózatok) kommunikációt alkalmaz.

3.2. A közlekedés iparban alkalmazott hálózatok

A következő pontokban, olyan technológiák kerülnek bemutatásra, amelyek jelen vannak a közlekedési iparban, annak valamely területén (közúti forgalomirányítás, járműgyártás, járműfedélzeti kommunikáció stb.) alkalmazzák azokat.

3.2.1. Ethernet

Az eredeti Ethernet, a Xerox által a 70-es években kifejlesztett kísérleti koaxiális kábelt használó hálózat volt, mely a CSMA/CD (Carrier Sense Multiple Access with Collision Detection) kommunikációs protokollon alapult. A kezdeti sikerek hatására, három cég (Digital Equipment Corporation (DIC), Inter, Xerox Corporation) összefogásával létrejött a 10Mb/s sebességű Ethernet specifikációja, majd ez alapján jött létre az IEEE 802.3 szabvány 1983-ban. Az azóta eltelt idő során több új típussal és módosítással bővült a 802.3-as szabványcsomag.

Működése a CSMA/CD, (vivőérzékeléses többszörös hozzáférés) alapján [11]:

Egy állomás, mielőtt adatokat küldene, belehallgat a csatornába. Ha az éppen nincs használat alatt, akkor elkezdi az üzenete küldését; ellenkező esetben véletlenszerűen meghatározott ideig vár. Az elküldött üzenet a csatornán keresztül minden állomáshoz eljut, azok az üzenetben lévő cím alapján eldöntik, hogy nekik szól-e (fogadják, illetve feldolgozzák), vagy nem (eldobják). Ütközés esetén a csatorna zavarásával jelzik a hibát, ezt követően az állomások véletlenszerű ideig várnak, kezdetben ez 0 vagy 1 időintervallumot jelent, majd ez az esetleges további ütközéseknél 1-el bővül. Ez alapján az n-edik ütközésnél 0 és 2n-1 időintervallum közötti számnak megfelelő időt kell várakozniuk.

Érthető módon, akkor a leggyorsabb a rendszer, ha alacsony az adatforgalom, mert kevesebb a várakozási idő.

Az Ethernet hálózatokban többféle kábeltípus is használható:

- árnyékolatlan csavart érpár
- vékony koaxiális kábel
- vastag koaxiális kábel
- üvegszálas kábel

3.2.2. PROFIBUS

A PROFIBUS (Process Field Bus) nyílt, gyártó független terepbusz kommunikációs szabvány, széleskörű alkalmazási területtel [12]. Ez a protokoll 1989-ben jött létre, a német kormány, valamint az automatikai termékekkel foglalkozó gyártók és intézetek közös munkájának köszönhetően. A technológia lehetővé teszi a különböző gyártók készülékeinek speciális interfészek nélküli kommunikációját. Több áramkörgyártó, főleg a Siemens implementálta ez a technológiát az általuk tervezett és gyártott áramköreikbe. A kommunikáció elsősorban rézkábel alkalmazásán alapul viszont nagy távolságoknál már optikai kábel alkalmazására van szükség [13].

A szabványnak három változata van, megjelenésük sorrendjében:

- PROFIBUS-FMS
- PROFIBUS -DP
- PROFIBUS -PA

PROFIBUS -FMS

Általános megoldást nyújt a cella szintű kommunikációs feladatok számára. Rugalmasságának és hatékonyságának köszönhetően széles körben felhasználhatók. Megoldást jelentenek a kiterjedt és bonyolult kommunikációs feladatok elvégzésénél. A Profibus-DP elődjének is lehet mondani.

PROFIBUS -DP

Elsősorban automatikus vezérlőrendszerek és elosztott I/O eszközök kommunikációjára fejlesztették ki. Nagysebességű és olcsó összeköttetésre optimalizált változat, amely akár párhuzamos kommunikációra is felhasználható. Jelenleg ez a legelterjedtebb PROFIBUS verzió

PROFIBUS -PA

Folyamatszabályozási feladatok elvégzésére fejlesztették ki. Alkalmazásával közös buszra lehet csatlakoztatni az érzékelőket és a beavatkozó egységeket, akár nagy megbízhatóságot igénylő környezetben is. Az alkalmazott két vezetékes technológia által a kommunikáción túl tápfeszültség továbbítására is alkalmas.

Mindhárom PROFIBUS változat ugyanazt a hozzáférési protokollt alkalmazza. A közeghozzáférés-vezérlés határozza meg az egyes eszközök adási lehetőségét, úgy hogy egy időben csak egy állomás adhat. A PROFIBUS megfelel a közeghozzáférés két fontosabb kitételének:

- komplex szabályozó rendszer esetében egy meghatározott időkeretben biztosítani kell az állomások számára az elegendő adási időt a kommunikációs feladataik ellátására.
- a komplex programozható vezérlők és azokhoz tartozó I/O eszközök közötti periodikus és valósidejű kapcsolatnak minél gyorsabbnak és egyszerűbbnek kell lenni.

Egy ilyen rendszer általános felépítése a következő: az aktív állomások (mester - master) logikai vezérjeles gyűrűt alkotnak (token ring). A vezérjelet (buszhozzáférési jog vagy token) az aktív állomások ciklikusan megkapják. Ha egy állomás megkapta a vezérjelet egy meghatározott ideig a kommunikációs feladatait. Mivel csak egy időben csak egy állomás adhat, így ütközések nem fordulhatnak elő.

3.2.3. CAN-BUS

Az utóbbi évek egyik legjobban elterjedt kommunikációs protokollja a CAN (Controller Area Network) [15] [16]. 1986-ban a Rober Bosch GmbH vállalat fejlesztette ki Németországban, elsősorban járműtechnikai célokra. Később, előnyös tulajdonságainak köszönhetően, általános célokra, más ipari területeken is el kezdték használni.

- Megbízhatóan alkalmazható a zajos, elektromágneses zavarokkal teli ipari környezetben
- Nagysebességet (1 Mbps) biztosít
- Félvezető támogatottsága, olcsó és kisméretű megvalósíthatóságot biztosít

A CAN hálózatban üzenetadáskor nincs konkrét címzett, ehelyett az üzenet, a csatornán keresztül, a buszra csatlakozott minden elemhez eljut, amelyek az üzenetben található azonosító alapján eldöntik, az üzenet vonatkozik-e rájuk. Az azonosító nem a csak az üzenet tartalmát jellemzi, hanem annak a prioritását is, ami fontos egy ilyen rendszerben, ahol több állomás verseng az adási jogért. 4 fajta üzenet típus létezik:

- Adathordozó üzenet (data frame): feladata az adatok továbbítása a rendszerben lévő állomások között
- Adatkérő üzenet (remote frame): egy állomás egy másik felé adatkéréssel is fordulhat
- Hiba üzenet (error frame): Buszhiba érzékelésekor hibaüzenet generálhatnak az elemek.
- Túlcsordulás üzenet (overload frame)

A technológia rugalmassága biztosítja az új állomások egyszerű hálózatra történő csatlakoztatását, anélkül, hogy a meglévő állomások szoftverét vagy hardverét módosítani kelljen. Ez természetesen csak akkor igaz, ha az új elem csak vevő, mert ha adó is, az általa küldött üzenetek azonosítója ismeretlen lesz a hálózat többi eleme számára.

A CAN kétfajta kerettípust ismer, melyek között az azonosító hosszában van eltérés. Az standard formátumú cím 11 bitből áll, míg a kiterjesztett, amelyre kapacitásbeli problémák miatt van szükség, 29 bitet tartalmaz.

Az esetlegesen felmerülő hibák detektálására három módszert alkalmaz:

- Cyclic Redundancy Check (CRC): redundáns ellenőrző-bitek segítségével tudja ellenőrizni a kapott információt.
- Keret ellenőrzés: az üzenetkeret felépítésének ismeretében, vizsgálja az eltéréseket
- ACK hibák: a vett keretet minden résztvevő elem nyugtázza az adó felé.

3.2.4. AS-Interface

Az AS-Interface (Actuator Sensor Interface) széles körben elterjedt nyílt terepi busz. Érzékelők (szenzorokat) és beavatkozók (aktorok), egy vezetéken keresztüli összeköttetését valósítja meg [16] [17]. Árnyékolatlan, kéterű vezetéket alkalmaznak az elemek összekötésére, amin keresztül biztosítják az adatforgalmat, valamint kisenergiájú eszközök esetén tápfeszültséget (24 V egyenáramot) is.

A teljes hálózat egy master állomást tartalmazhat, ami a lekérdezéseket végzi. A rendszer automatikusan érzékeli az újonnan becsatlakoztatott eszközöket, így nincs szükség konfigurációra és alkalmazás-specifikus szoftverre sem. A becsatlakoztatott slaveveknek 2 fajtája van intelligens eszközök (pl. kapcsolók), és modulok, melyek a hagyományos bináris eszközök csatlakoztatására szolgálnak.

A rendszer jellemzői:

- Gyors
- Üzemi környezetben is megbízható
- Olcsó
- Könnyen továbbfejleszthető rendszer
- Kis helyigényű, könnyen bővíthető
- Gyors hibaelhárítási lehetőség.

3.3. Protokollok a forgalomirányító központ és a terepi gépek közötti kommunikációhoz

A központ és a csomóponti készülékek közötti kommunikáció, csakis valamilyen, a szabályokat lefektető protokoll keretében történhet. A különböző protokollok eleinte feladat-specifikusak voltak, vagyis konkrét, megszabott feladatkör elvégzésére lettek létrehozva. Erre példa a Siemens BEFA protokolljai, melyek igen széles körben elterjedtek, még ma is használják azokat.

Később, a hálózati technológiák fejlődésével együtt bővültek a lehetőségek, így mód nyílt általános-célú protokollok alkalmazására, mint például TCP/IP. Erre épül rá a több gyártó összefogásával készült protokoll is: az OCIT.

3.3.1. BEFA

A BEFA protokollokat a Siemens fejlesztette ki. A központ és a terepi berendezések közötti kommunikációnál terjedt el. A BEFA protokoll-család tagjai: BEFA 5/51, 8/81, 10, 12, 15, 16, 17. A legelterjedtebbek a 12, 15, 16 [18]

3.3.1.1. BEFA 12

Frekvencia és idő multiplex rendszer. 20 csatorna megkülönböztetésére képes 20 frekvencia-tartományban (TST20). Az információtovábbítás soros átvitellel történik ezeken, a kommunikációs csatornákon. Megbízható, köszönhetően a megvalósított két érpáros rendszernek, így a mai napig elterjedt. Hátránya, hogy kezd elavulttá válni, nem mindig felel meg az elvárt átviteli sebességnek. A gazdaságosság jegyében, az elemeket (maximum 5 darabot) egy közös láncra lehet felfűzni, így csökkenthetők a kábelezésre szánt kiadások, viszont így egy elem hibája folytán, az egész hálózat leállhat.

3.3.1.2. BEFA 15 és BEFA 16

A BEFA 15 működésének alapja a modern táviratos rendszernek mondható. Az egyes elemek egy közös buszra csatlakoznak, melyen keresztül címzett táviratokkal érhetőek el. Előnye a stabil, megbízható működés mellett a nagy átviteli sebesség. Alkalmazásával több információ cserélődhet a központ és a terepi berendezés között, így jelzőcsoport távvezérlésre is alkalmas.

A BEFA 16 esetében már igazi buszrendszert alkalmaznak, így elődéhez képest nagyobb átviteli sebességre képes.

3.3.2. OCIT

Az utóbbi években, a közúti közlekedési forgalomirányításban, az OCIT protokoll kezd egyre elterjedtebbé válni. Az OCIT egy levédett márkanév. Teljes neve az Open Communication Interface for Road Traffic Control Systems, vagyis nyílt kommunikációs interfész a közúti forgalomirányító rendszerek számára [19].

Az OCIT protokollt Németországban fejlesztették ki, öt cég (Dambach, Siemens, Signalbau Huber, Stoye, Stührenberg) együttműködésével. Így a technológia elsősorban azokban az országokban kerül alkalmazásba, ahol az előbb említett cégek működnek.

Ez a protokoll kizárólag ipari felhasználású, a forgalomirányítás területén alkalmazzák. Egyik legfontosabb tulajdonsága, hogy általa a forgalomtechnikai elemeket, vagyis az irányító berendezéseket, központi elemeket egy közös hálózatba lehet foglalni.

Az OCIT rendszer több elemet is magába foglal:

- Rendszer-architektúra
- Szabályok
- OCIT-protokoll
- Funkciók
- Átviteli protokoll

Az OCIT-architektúra három interfészterületet különböztet meg a közlekedésirányításban:

- OCIT-belső állomás: a központi elemek közötti szabványosított interfészek
- OCIT-külső állomás: a központi elemek és a terepi berendezések közötti szabványosított interfészek
- OCIT-LED: A terepi berendezések és a jelzőkészülékek közötti elektromos interfész.

Az OCIT adatátviteli technika a szabványos TCP/IP kapcsolaton alapszik, tehát megbízható kommunikáció valósítható meg.

A TCP/IP protokollkészlet két legfontosabb protokollja

- TCP (Transmission Control Protocol): a szállítási rétegben működő, megbízható pont-pont csatornát biztosító protokoll. Feladatai közé tartozik az üzenetek széttördelése és összeállítása, az eltűnt csomagok újraadása és a részek helyes sorrendjének visszaállítása.
- IP (Internet Protocol): a hálózati rétegben működő protokoll. Fő feladata, hogy az egyes adatcsomagokat a megfelelő útvonalon keresztül, eljuttassa a címzettnek.

Az OCIT rendszernek saját adatátviteli protokollja van, a BTPPL (Basic Transport Paket Protokoll Layer), amely együtt működik az internetes szabvánnyal. A BTPPL adatfolyam egy magasabb, illetve egy alacsonyabb prioritású csatornákból áll. Az előbbit elsősorban kapcsolási parancsok, illetve jelentések küldésére használják, míg az utóbbin a működés szempontjából kevésbé fontos adatok cseréje megy végbe.

3.4. Fejlesztési tendenciák

A jelenlegi fejlesztési irányok, a közlekedési irányítólámpák fejlődését, tudásuknak bővülését mutatják, mondhatni a forgalomirányító berendezések „kárára". A cél, a jelenlegi, a jelzőfejek és a berendezés kapcsoló kártyája közötti pont-pont alapú összeköttetés helyett, buszos hálózat kialakítása, melyen keresztül a felfűzött jelzőfejek táviratokkal kommunikálhatnának a berendezéssel. Ezen túlmenően, a terepi gépekből elhagynák a kapcsolókártyát, mivel a lámpakapcsolások helyben, a jelzőfejbe épített lámpakapcsoló eszközben történne. A kialakuló intelligens egységeknek a következő feladatokat kell ellátniuk:

- Kommunikáció biztosítása a csomóponti berendezéssel
- Lámpakapcsolás elvégzése
- Izzóellenőrzés

A koncepció megvalósításával a kábelezési költségek csökkenésére, valamint az új egységek kialakítása miatt, többlet kiadásokra kell számítani.

Bár a lámpakapcsolás és az állapotellenőrzés a jelzőfejnél történik, a közbensőidők és a tiltott irányok ellenőrzése továbbra is a forgalomirányító berendezésben maradna. Mivel a két rendszerelem hálózaton keresztül tud kommunikálni, így az újonnan felmerülő biztonsági aggályok miatt, mind a berendezést és mind a jelzőfejeket egy magasabb szintű, kommunikációt ellenőrző funkciókkal kell kibővíteni, melyek szükség esetén beavatkoznának.

Az előbbi gondolatsort folytatva, a jövőben már elképzelhetők lennének olyan forgalomirányítási struktúrák, melyekből hiányoznának a terepi berendezések, vagy legalább-

is megváltozott feladatkörrel szerepelnének. Az egyes jelzőfejek vezérlése a központból vagy alközpontból történne, mivel minden, az adott csomópontra vonatkozó információ ott lenne eltárolva. Az új rendszer alkalmazása számos előnnyel járna:

- Költségcsökkenés
    - Forgalomirányító berendezések elhagyásából
    - Nincs szükség komolyabb kábelezésre, mivel a rendszer interneten keresztül működve, így az összeköttetés a központ és a jelzőfej között, már eleve meg van
    - Kevésbé költséges a kiépítés és üzemeltetés
- A hálózat könnyű bővíthetősége
- Egyszerűbb rendszerkarbantartás

Komolyabb hibák esetére, viszont továbbra is szükség lenne egy helyi beavatkozó egységre, ami egyben hálózati kapcsolóként (biztosítva az internetre való kapcsolódást), és a csomóponti elemek tápegységeként is üzemelne [25].

4. A rendszer megtervezése

4.1. A tervezett rendszer általános leírása

4.1.1. A rendszer jellemzői

A megvalósítandó rendszer alapvetően centrális (központos) struktúrájú, tehát a főbb számításokat egy központi számítógép fogja elvégezni Ennek köszönhetően, a csomóponti berendezések feladatköre, a központtal való kommunikáción túl, a detektoradatok összegyűjtésére, a kapott, kiszámított adatok alapján történő programmódosításon keresztüli lámpavezérlésre és az ellenőrzési feladatok (program helyességének ellenőrzése a közbensőidő és tiltási mátrix alapján, lámpaellenőrzés) elvégzésére szűkül. A cél a csomópont forgalomfüggő irányítása, aminek szükséges feltétele, hogy képet kapjunk az aktuális forgalmi helyzetről, melynek legalapvetőbb eszköze a hurokdetektorok alkalmazása. A pontos számításokhoz és az aktuális forgalmi helyzethez leginkább illeszkedő irányítás létrehozásához, a jelenlegi rendszer, más forgalomfüggő rendszerhez hasonlóan, a szabályozott csomóponti irányok, sávok minél alaposabb detektorokkal való lefedettséget igényli. Minimum követelménynek az egy fázisba tartozó irányok közül legalább egynek a forgalmát kell mérni.

A kliensek és a szerver között socket-alapú kommunikáció van kiépítve. Rövid összefoglalás a socket-alapú kapcsolatokról:

- A szervernek van egy kitüntetett portja, amin keresztül fogadja a kliensek kapcsolódási kéréseit.
- A klienseknek ismerniük kell a szerver hálózati azonosítóját, valamint, az előbb említett kitüntetett port számát, amire a kapcsolódási kérelmet küldi. A kliensek
- A kapcsolati kérés elfogadását követően, a szerver lefoglal egy új portot a klienssel való kapcsolatra. Erre azért van szükség, hogy a kérések figyelése folytatódhasson, amíg a kliens kapcsolódik a szerverre.

Szervernek, vagyis központnak, jelen esetben egy asztali számítógép van beállítva, ezen fut az irányításhoz szükséges programrendszer. Mai átlagos számítógépek már elég gyorsak, a számítási feladatok ellátásához, viszont tanácsosabb egy nagyobb teljesítményű, folyamatos üzemre gyártott számítógép alkalmazása.

Az irányító központ szoftverei MATLAB programnyelven/fejlesztői környezetben készültek: Az adott csomópontok működését szabályzó számítások, a terepi berendezésekkel történő kapcsolat kialakítása, és egy felhasználói alkalmazás, mely grafikus felületen keresztül biztosítja a beavatkozási lehetőséget a szabályozási folyamatba és az elmentett adatokhoz való hozzáférést. Számtalan programnyelv létezik, ami alkalmas lehet hasonló feladatok megoldására. A MATLAB mellett való döntés, annak, a feladat szempontjából előnyös tulajdonságai miatt történt:

- A MATLAB-ot eleve komplex számítások elvégzésére és algoritmusok könnyű megvalósítására hozták létre, így a tervezett rendszerhez szükséges kalkulációk és eljárások egyszerűen létrehozhatók és gyorsan elvégezhetők és ábrázolhatók.
- Egyszerű szintaktikával rendelkezik, sok nyelvi elem a JAVA programozási nyelvből lett át véve
- Képes grafikus interfészek létrehozására
- Teljes mértékben képes a szükséges feladatok megvalósítására (fájlkezelés, hálózati kommunikáció, adatábrázolás)

A létrehozott központi irányító szoftver jellemzői:

Alapvetően két nagyobb egységre bontható:

- A szerver feladatokat ellátó rész: számítások elvégzése, kapcsolat kiépítése, adatok archiválása
- Grafikus felhasználói kezelő felület, mely a korlátozó feltételek módosítását és a mentésre került adatokhoz való hozzáférést biztosítja.

Ezek a részek ténylegesen is el vannak különítve egymástól. Ennek egyrészt biztonsági oka van, ugyanis így a felhasználói alkalmazás kritikus hibája esetén, a szerverszolgáltatás nem áll le, mivel az irányításhoz szükséges adatok külön állományokban vannak tárolva. A másik ok a részek szeparációjára, a számítógép rendszer erőforrásgazdálkodás. Így meg lehet szabni, hogy a kevésbé fontos grafikus felhasználói programrész prioritása alacsonyabb legyen (kevesebb processzoridőt kapjon), a fontosabb szerver programétól. Ezen kívül lényeges, ha nincs szükség a korlátozó beavatkozások szabályozására és adatlekérdezésre, ábrázolásra, akkor egyszerűen kikapcsolhatjuk a felhasználói programot, csökkentve a processzor terheltségét és a felhasznált memória nagyságát.

A szabályozás, a korlátozó adatok figyelembevételével történik. Ezek az adott fázisra vonatkozó kioszthatóó zöldidők legkisebb és legnagyobb értékei. Ehhez egy kis pontosítás szükségeltetik: a fázisterv készítésekor a megállapított fázis zöldidőket az adott fázisba tartozó, legnagyobb forgalommal rendelkező, járműves irány kapja meg. Általában a többi irány ezzel az értékkel megegyező, vagy kisebb értéket kaphat. Viszont, a rendszer arra is fel van készítve, hogy egy irány a megállapított zöldidőtől hosszabb szabadjelzést kapjon. Ez csakis akkor lehetséges, ha a következő fázis összes eleme ezt megengedi (pl.: ha a szóban forgó irány, két fázisba is tartozik). A rendszer állandó fázissorrendekkel operál, ugyanis, erre a lehetőségre, jelen kialakításban, csak elvétve lenne csak szükség.

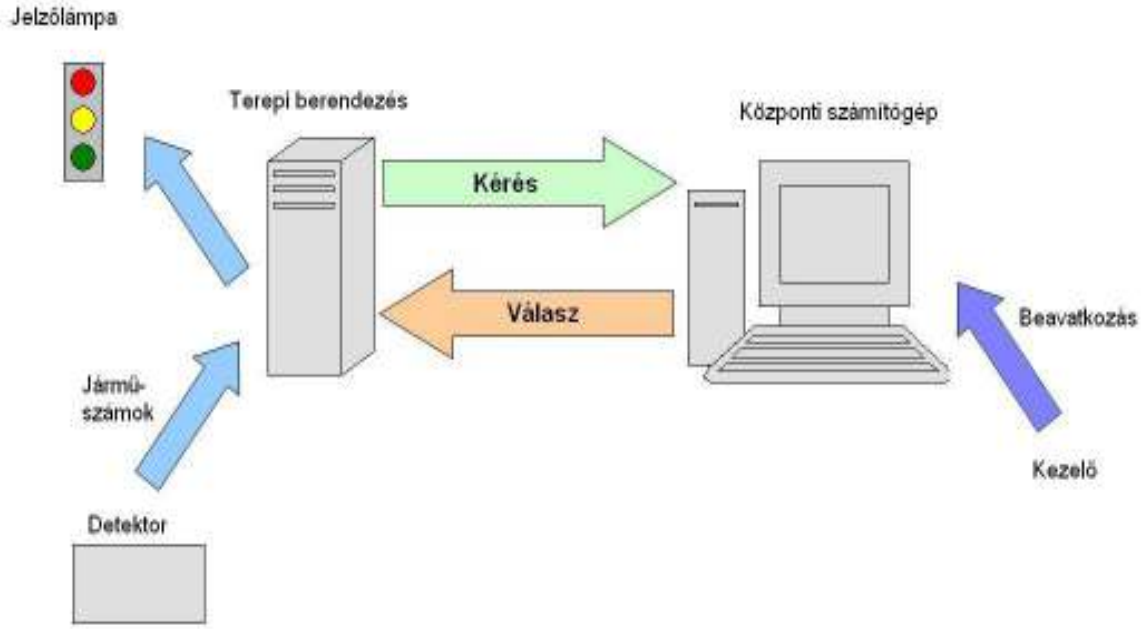
Ahhoz, hogy a forgalomfüggő irányítás még inkább megfeleljen az igényeknek, fel lett készítve a periódusidő manipulálására is. Nagyobb forgalom esetén megnöveli a periódus időt, kisebb forgalom esetén, pedig csökkenti. A rendszer által alkalmazott periódusértékek 50 és 100 közötti, 10-zel osztható értéket vehetnek fel. A folyamat leírása a 4.1.3-es fejezetben olvasható.

A kliens oldali gépeknek, ACTROS VTC 3000 forgalomirányító berendezések lesznek beállítva. Természetesen a rendszer gyakorlati alkalmazásánál az ACTROS gépek kliensként történő alkalmazása, nem lenne gazdaságos, mert annak tudása jóval meghalad-

ja az általunk szükségesét, ugyanis is minden, az irányításhoz szükséges számítást a központ végez Ezek egyik legfontosabb előnye, hogy programozása JAVA nyelven történik. Ez a programnyelv, más nyelvek ismeretében, viszonylag könnyen és rugalmasan használható. Az új funkciókat ellátó kliens program, egy már meglévő programrendszert egészít ki. Többletfunkciók:

- Adatgyűjtés
- Adatcsere a központtal
- Módosítások életbeléptetése

A rendszer elemei és a közöttük zajló folyamatok az alábbi ábrán (5. ábra) is jól végigkövethetők:



5. ábra: A rendszer elemei

4.1.2. A rendszer működésének felvázolása

A kezelő, a szerver felhasználói interfészen keresztül, különböző paraméterek megadásával (jelen esetben zöld idők alsó és felső korlátainak beállításával) s a központi program működését, ezáltal az egész szabályozási folyamatot. A központi szoftver indítása után a kliensek kérésére vár.

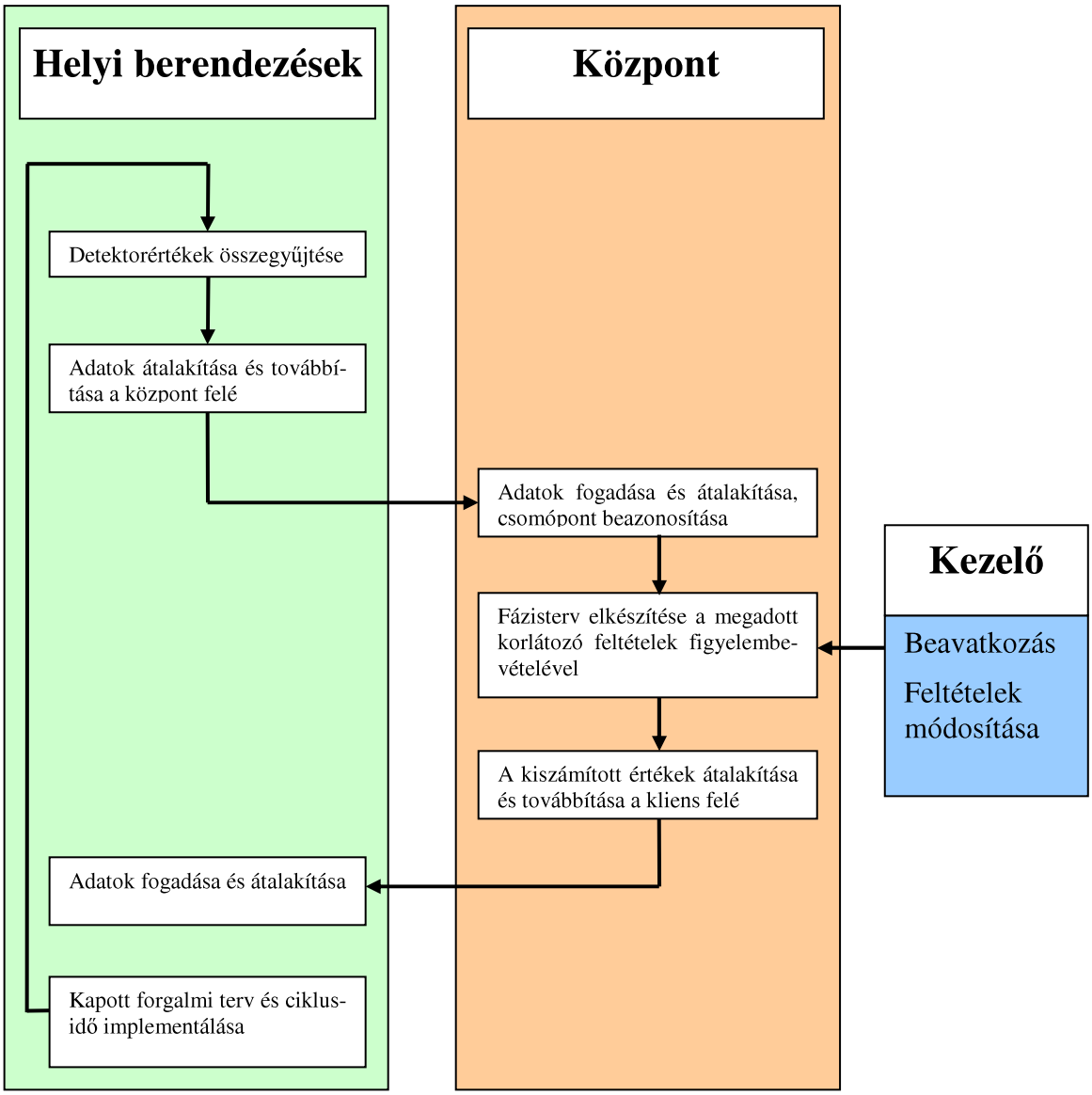
A kliens oldalon található ACTROS készülékek, a jelen példában, a megszokotthoz képest kevesebb feladatot látnak el. Bizonyos időközönként lekérik a hozzájuk rendelt csomópontok detektor-értékeit, ami végül is a legutóbbi lekérdezés óta, a mérőberende-

zéseken áthaladt járművek száma. Ezeket megfelelő formában átküldi a központi számítógépnek.

A szerver-oldali számítógép miután fogadta az adatokat, a kliens hálózati azonosítója (IP cím) alapján, egy adatbázis segítségével beazonosítja a klienst és betölti az ahhoz tartozó, a szabályozáshoz szükséges adatokat. Ezt követően, a következő pontokban részletezett algoritmusok alapján kiszámítja, az új, az aktuális forgalmi viszonyoknak legmegfelelőbb ciklusidőt és fázistervet, majd megfelelő formátumban továbbítja azokat, a terepi berendezés felé.

A forgalomirányító készülék a kapott ciklusidő és forgalmi terv, alapján módosítja a jelenleg futó programot.

A teljes folyamat a 6. ábrán is látható.



6. ábra: A rendszer működésének folyamata

4.1.3. Az periódusidő számítás

A rendszer megvalósításakor az egyik célkitűzés az volt, hogy a forgalomhoz való alkalmazkodás terjedjen ki a berendezések a szükséges periódusidő (ciklusidő) kiszámítására is. Jelen megvalósításban, a számításokhoz több statikus és dinamikus adat szükségeltetik.

Statikus adatok:

- Detektorok, mely jelzőcsoportokhoz tartoznak
- A jelzőcsoportok mely fázisba vannak sorolva

Az adatbázisból, a fent leírt adatok lekérése után, a detektorok fázisokhoz rendelése innen már könnyen megy, amire a számításban lesz szükség.

Dinamikus, változó adatok:

- Aktuális periódus idő
- A detektorok által mért adatok

A fenti adatok alapján, először a fázisokhoz tartozó, mértékadó járműszám megállapítása kerül sor. A mértékadó járműszám, az aktuális fázishoz tartozó irányokon, a jelenleg mért legnagyobb járműszám. Így a mértékadó irányok között lévő közbensők lesznek a fázisok közötti közbenső idők is.

Az új ciklust megállapító algoritmus alapja a következő összefüggések:

$P = \sqrt{\frac{120 \cdot \sum_{i=1}^{fáziszám} t_{k\_i}}{1-Y}}$, ahol a $\sum_{i=1}^{fáziszám} t_{k\_i}$ a fázisok közötti összes közbensőidőt jelenti.

$Y = \sum_{i=1}^{fázisszám} \frac{M_i}{M_{\max}}$, ahol az M_i az adott fázis mértékadó forgalom (jármű/óra), az $M_{\max}$ pedig 1800 jármű/óra, vagyis egy sáv által biztosítható legnagyobb járműnagyság.

Jelen megvalósításban, az adott fázisok mértékadó forgalmát az adott periódusban mért detektor adatok alapján számoljuk ki.

$M_i = \frac{x_{\max\_i} \cdot 60}{P_{aktuális}} \cdot 60$, ahol $x_{\max\_i}$ az adott fázis sávjaiban mért járműszámok legnagyobbika, vagyis a mértékadó járműszám. A képlet magyarázata: ha a mért járműszámot megszorozzuk a $60/P_{aktuális}$ –sal, megkapjuk az egy percre vonatkoztatott járműszámot. Ezt, ha ismételten 60-nal beszorozzuk, megkapjuk az órai forgalom becsült értékét.

A jelen rendszer által alkalmazott periódus idő értékek 50 és 100 másodperc között vehetnek fel, tízzel osztható időértéket (50, 60, 70, 80, 90, 100), mivel felesleges és negatív következményei is lehetnek a ciklusidők másodperces változtatgatásainak. A maximálisan megadható ciklusidő 100 másodperc, annak ellenére, hogy a szakirodalom 120 másodpercet is meg enged. Erre a kikötésre elsősorban a közlekedők miatt került sor, mivel a nagy ciklusidő hosszú tiltójelzéssel is jár, ami már nem túl kényelmes a türelmüket elvesztettek számára.

Levonható, hogy a rendszer megköveteli az adott csomópont, annak sávjainak forgalmát mérő, detektorokkal való „lefedettségét". Minimum követelmény a fázisonkénti 1 mérőberendezés kihelyezése.

4.1.4. A fázisok zöldidőit optimalizáló algoritmus leírása

A jelen esetben készített algoritmus egészen egyszerű: az adott csomópontoknál, az egyes fázisokhoz tartozó zöld idők alsó és felső korlátait tartalmazó tömbök, az összes hasznos idő és a kapott járműszám alapján történik az irányításhoz szükséges paraméterek, vagyis a fázisok zöldidőinek kiszámítása.

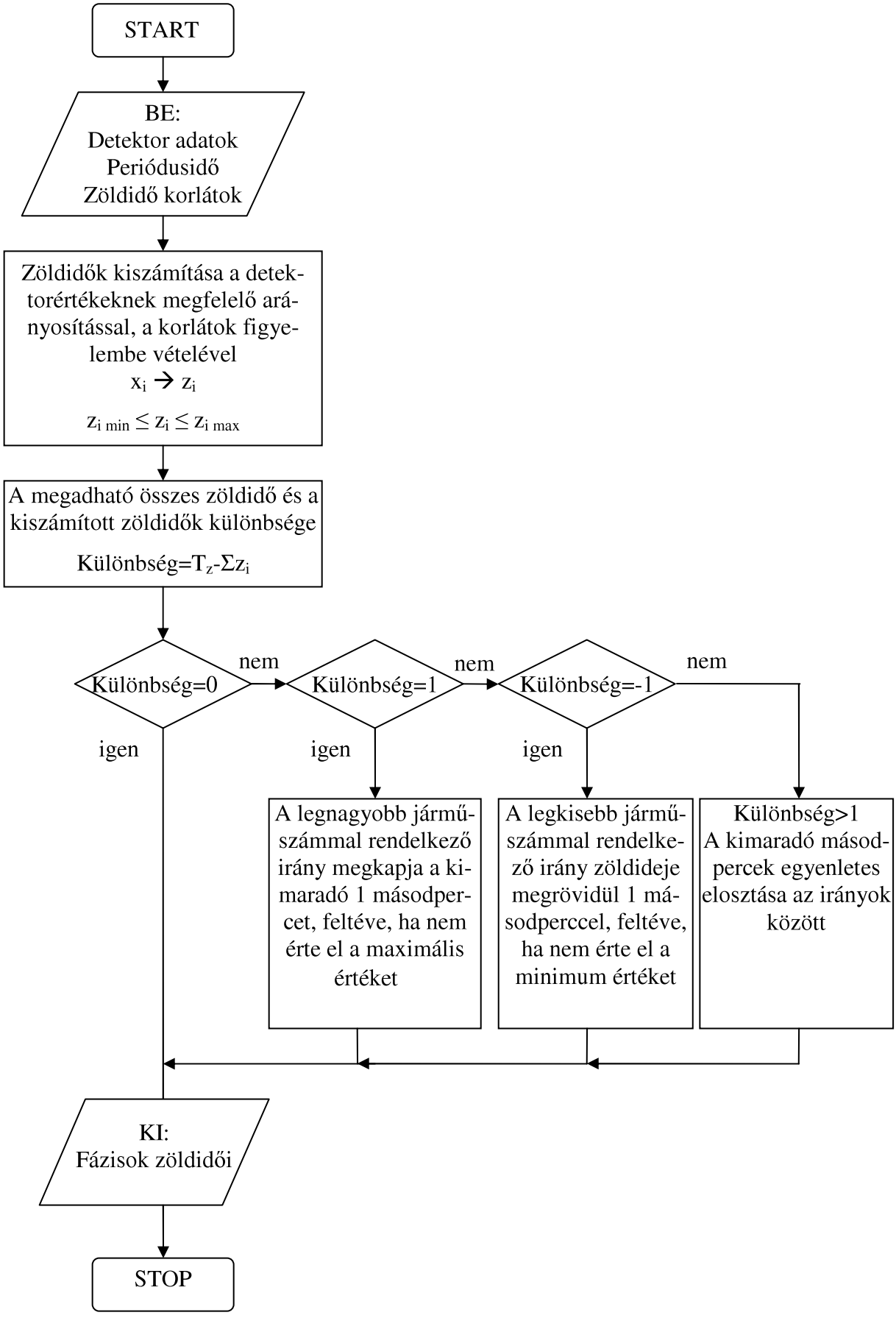
Először, a hasznos zöldidőt (ami az újonnan megállapított periódusidő és a fázisok mértékadó irányai közti közbensőidők különbsége) a kapott detektor adatoknak, vagyis a járműszámoknak megfelelően arányosan elosztjuk. Ezt intervallum-vizsgálat követi: a számított zöld idő érték benne van-e a megadott minimum és maximum értékek közötti megengedhető szabad jelzés tartományban.

Ha nem, tehát kisebb, mint a minimum vagy nagyobb, mint a maximum, akkor annak a fázisnak a zöld ideje rendre megkapja az ott megengedhető legkisebb vagy legnagyobb értéket. Ezt, ismételten arányosítás követi, de itt már a hasznos idő és az előbb meghatározott zöld idők különbségét kell leosztani a maradék, mértékadó detektor adatok alapján. Ez a folyamat egészen addig ismétlődik, amíg minden kapott számadat a korlátokon belül nem helyezkedik el.

Az esetek nagy részében ez így elég is lenne, de ritkán előfordulhatnak eltérések: a hasznos idő nem egyenlő az összes zöld idő összegével (T_z-$\Sigma z_i \neq 0$), ami az arányosítás során alkalmazott kerekítéseknek köszönhető. Abban az esetben, ha az eltérés +1, vagyis a kiszámított összes szabad jelzés hossz kisebb, mint a megengedhető zöld idők összege, akkor a kimaradó 1 másodpercet, a legnagyobb járműszámmal bíró irány kapja meg (természetesen a korlátokat is figyelembe véve). -1 eltérésnél is hasonlóan járunk el, ott a legkisebb járműszámú irány zöld idejéből kerül levonásra egy másodperc.

Meg kell még említeni, olyan szélsőséges esetet is, amikor olyan alacsony a járműforgalom, hogy több irány is csak a minimum zöld időt kapja meg és az eljárás végén, és így, több másodperc is lehet a hasznos idő és a kiosztott zöldidők közti differencia. A megoldás egyszerű: a maradék másodpercek egyesével leosztásra kerülnek az egyes irányok között, mindaddig, amíg el nem fogynak. Magától értetődően itt is figyelni kell, hogy ne haladja meg az adott irányra vonatkozó felső korlátot.

Az előbb ismertetett algoritmus a 7. ábrán is végig követhető.



7. ábra: A hasznos zöldidők szétosztása

4.1.5. Az új fázisterv elkészítése

Az új rendszer egyik legfontosabb része az aktuális, vagyis a jelen helyzethez leginkább megfelelő fázisterv elkészítése. Ezen feladat megoldására létrehozott algoritmus itt is, több statikus és dinamikus adatot használ fel:

- Statikus adatok
    - Közbenső idők
    - Jelzőcsoportok típusai (jármű, gyalogos, villamos)
    - Fázisok. Érthető módon a fázisterv elkészítéséhez szükséges az egyes irányok fázisokba sorolása
    - Adott irányokhoz tartozó detektorok ismerete
    - Offset: az egyes csomópontok szabadjelzéseinek összehangoláshoz szükséges idő érték.
- Dinamikus adatok
    - Periódusidő
    - Mértékadó irányok (gépjárműves, tehát nem gyalogos, nem kerékpáros csomóponti mozgások). Nem minden esetben állandó, mivel egy fázisba több, detektorral mért irány is tartozhat, közülük a mértékadó irány az éppen legnagyobb járműszámmal bíró irány lesz.
    - A fázisokhoz, pontosabban a mértékadó irányokhoz megállapított zöld idők

A fázisterv készítésének menete:

1. Először a fázisok mértékadó irányai kerülnek be a tervbe, a következő módon: az első fázis mértékadó irányának szabad jelzése a ciklusidő fix, mindig állandó időpontjától kezdődik (jelen esetben ez a 3. másodperc), majd ehhez adódik a hozzátartozó zöldidő hossza, így ezáltal megkapjuk a szabadjelzés befejező időpontját. Ehhez, ha hozzáadjuk az aktuális és a következő fázis mértékadóirányai közti közbenső időt, megkapjuk a következő fázis mértékadó irányának a zöldidejének kezdetét. Az itt megállapított kezdési és befejezési időpontok a számítás további menete alatt, nem fognak változni, ugyanis a többi jelzőcsoport jelzéstervét ezekhez fogjuk viszonyítani.
2. Ezt követően következnek a fázisok nem-gyalogos csomóponti mozgásaira vonatkozó zöldidők megállapítása. Az adott irány zöldidejének kezdetét az előző fázis irányainak és a köztük levő közbensőidő határozza meg, míg a végét azonos elgondolás alapján, a következő fázis, már kiszámított irányainak kezdete és a közöttük lévő közbenső idő alapján fog megállapításra kerülni. A számítás során a rendszer azt is figyelembe veszi, ha egy irány két fázisba is tartozhat.
3. A járműves csomóponti mozgások megállapítása után következnek a gyalogos szabad jelzések kezdeteinek és befejezéseinek megállapítása. Az egész folyamat a 2. pontban leírtak alapján történik. Erre a „megkü-

lönböztetésre” azért volt szükség, hogy a járműves irányokra vonatkozó számítások ne függjenek a gyalogosforgalométól, így kapva, igaz csak minimálisan több zöldidőt.

4. Ha a csomópont a gyalogosforgalomra figyelmeztető villogókkal van ellátva, akkor azok kapcsolási időpontjainak számítása a hozzájuk rendelt gyalogos jelzőcsoportok zöldidő adatai alapján fog történni. Általában, a villogók 2 másodperccel a gyalogos forgalom megindítása előtt kezdenek el jelezni, és ezt egészen a tiltó jelzést követő 1 másodpercig folytatják.
5. Végül az egész terv, az offset változóban szereplő másodpercértékkel való eltolása következik, amire a csomópontok összehangolása miatt van szükség.

Az eredményül kapott ciklusidő végül, összehasonlításra kerül a hálózat, illetve a részhálózat más csomópontjainál megállapított ciklusidőkkel, és a legnagyobb fog alkalmazásra kerülni. Így, az irányított területen belül minden jelzőlámpával irányított kereszteződésnél meg fog egyezni a periódusidő. Erre elsősorban a közlekedés harmonizálásáért, és zöldhullámok kialakítása végett van szükség.

4.2. Programozáshoz szükséges eszközök

4.2.1. MATLAB

A szerver-oldali alkalmazás, beleértve a felhasználói interfészt (GUI), a kliensekkel történő kapcsolatkezelést és az egyszerűbb irányítási algoritmust, MATLAB fejlesztői rendszerben lettek megírva.

A MATLAB, a The MathWorks vállalat által, C és JAVA nyelvben kifejlesztett, tudományos és műszaki számítások elvégzését szolgáló környezet és program nyelv [20]. A MATLAB negyedik generációs programnyelvek közé tartozik. Ezek, jellemzően egy adott feladatkörre készültek, kevés nyelvi elemmel operálnak, egyszerű szintaktikával rendelkeznek, vagyis jelen esetben ez azt jelenti, hogy nem szükséges komolyabb programozói tudás a felhasználó részéről, az egyes matematikai leírások, kifejezések, műveletek könnyebben és egyszerűbben megfogalmazhatók. Viszont, összetettebb munkavégzésnél már legalább alapszintű programozói ismeretek szükségeltetnek.

MATLAB név a Matrix Labratory rövidítése. Világszinten ismert szoftver, mely nagy segítséget jelent a különböző szakterületeken (pl.: matematika, szabályozástechnika, stb). A különböző eszköztárak hozzáadásával növelhető a felhasználhatósági köre. Ezek az eszköztárak konstans adatokat és függvényeket tartalmaznak az adott témakörre vonatkozóan.

Főbb jellemzői: interaktivitás mellett képes:

- mátrixműveleteket végezni
- adat és függvényábrázolásra

- vizuális megjelenítésre
- algoritmusok végrehajtására
- valamint felhasználói interfészek kialakítására.

4.2.2. JAVA

Sun Microsystems által kifejlesztett objektumorientált nyelv, mely széles körben felhasználható, általános célú, tehát nem korlátozódik bizonyos szakterületekre [21]. Legnagyobb előnye a platformfüggetlenség, ami azt jelenti a JAVA nyelven megírt programok ugyanúgy fognak lefutni a különböző hardvereken. Ez úgy valósulhat meg, hogy a megírt programot (java kiterjesztésű) egy közbülső kódra, úgy nevezett bájtkódra fordítunk egy JAVA Compiler (fordítóprogram) segítségével, és ezt a platformfüggetlen kódot fogja értelmezni és futtatni, tehát gépi kódra fordítani a JAVA virtuális gép.

A JAVA nyelv szintaxisát a C és C++ nyelvektől örökölte. Erősen objektum-orientált nyelv. Alapja az osztályok, melyek az adatok és viselkedések összességéből álló példányok sablonját adják meg.

Főbb tulajdonságai [22]:

- Egyszerű: szintaktikája nagyban hasonlít a C++ nyelvéhez, attól viszont egyszerűbb is.
- Objektumorientált
- Előfordított
- Robosztus: kiszűri a programozási hibákat
- Biztonságos: megakadályozza a kártevő programok rendszerünkbe kerülését
- Hordozható: a JAVA programozás előnye, hogy a bájtkódra fordított program, más hardver, illetve szoftver struktúrájú környezetben ugyanúgy fog lefutni, köszönhetően a JAVA virtuális gépének.
- Semleges architektúrájú: az előző pontban leírtak alapján, a JAVA program nem függ a szoftver, hardver architektúráktól
- Többszálú: Lehetővé teszi, hogy az egyes programrészek párhuzamosan futhassanak le, ezzel is növelve a rendszerkihasználtságot és a feladat elvégzésének gyorsaságát.
- Dinamikus

Fejlesztői környezet

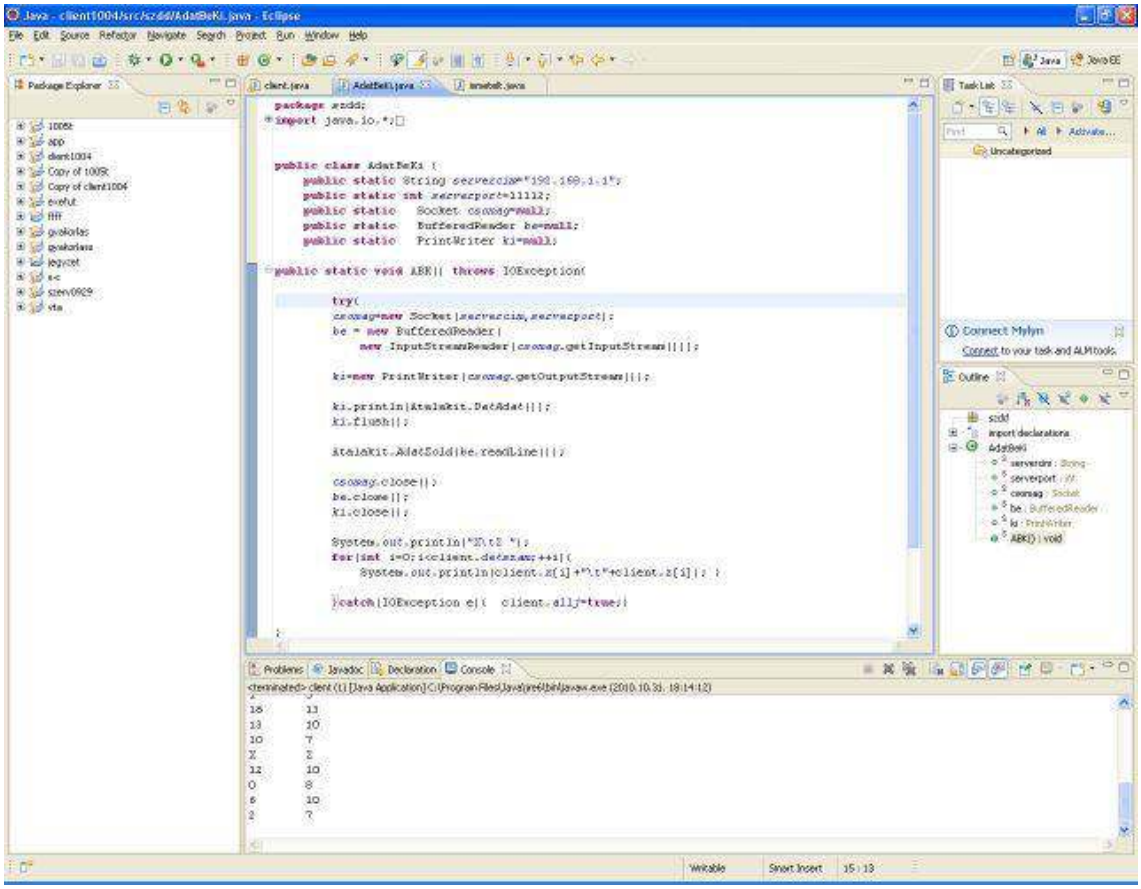
A JAVA nyelven történő programfejlesztésre, több erre alkalmas fejlesztői környezet létezik, melyek a különféle funkcióikkal (pl.: automatikus szintaxis-ellenőrzés) megkönnyítik a programozói munkát:

- JCreator
- NetBeans
- Eclipse

Jelen rendszer fejlesztésénél az utóbbi került alkalmazásra.

Eclipse

Platformfüggetlen (szinte az összes operációsrendszerhez van valamilyen változata), nyílt forráskódú, több programozási nyelvet támogató (JAVA, C++, PHP, stb.) szoftverkeretrendszer [23]. Számtalan kiegészítő eszközeinek köszönhetően hatékony és felhasználóbarát környezetet biztosít a felhasználók számára (6. ábra). Az IBM fejlesztette ki. Használata ingyenes, egészen addig, amíg kereskedelmi forgalomra szánt szoftvert nem készítünk, mert azután jogdíjat kell fizetni.



8. ábra: Az Eclipse kezelőfelülete

4.2.3. Az ACTROS programozása

A szakdolgozat keretében készülő központi forgalomirányító szoftverrel történő kapcsolat létrehozásához, a szabályozáshoz szükséges adatforgalomhoz, valamint a változtatások életbeléptetéséhez, a forgalomirányító berendezés (ACTROS) jelenlegi működésének módosítására és új funkciókkal való bővítésére van szükség. Ehhez, azonban elengedhetetlen a készülék programtechnikai működésének ismerete.

Az ACTROS programozása Java nyelven történik. Mint már szó volt róla, ez a nyelv objektumorientált, működésének alapja az objektum, és ez az ACTROS esetében ki is van használva: a különböző rendszerelemek, különböző objektumokként vannak szerepeltetve. Az objektumokat valamely közös tulajdonságuknál fogva osztályokba lehet sorolni, így többek között külön osztálya van a csomópontoknak, jelzőlámpáknak és jelzőcsoportoknak, detektoroknak, a berendezés egyes hardvereinek (IO kártya), a forgalomtechnikai programoknak (ez lehet fix, illetve forgalomfüggő), stb. Az ACTROS esetében, az adott csomópontra vonatkozóan a VT-Package tartalmazza ezeket az osztályokat és az egyéb forgalomszabályzáshoz szükséges programrészeket.

A VT-Package osztályai:

- Var.java: globális változók osztálya
- Init.Java: inicializálás osztálya
- FixProgX.java: X. számú programtervnek osztálya
- Beprog.java, Kiprog.java, SVProg.java: A be és kikapcsoló, valamint a sárgavillogó programok osztályai
- ForgfProg.java: a forgalomfüggő program osztálya
- AltalanosResz.java: metódusa minden (belső) ciklusba lefut. Feladata a gyalogos gombok alaphelyzetbe állítása, valamint detektorhiba esetén, fixprogramra való átállítás.
- LmpHb.java, LmpHb230v.java: a fénypontok hardverszintű azonosításáért felelő osztályok
- K.java: A magyar programozók munkájának megkönnyítésére szolgál, a benne foglalt metódusok segítségével

A berendezés üzeme

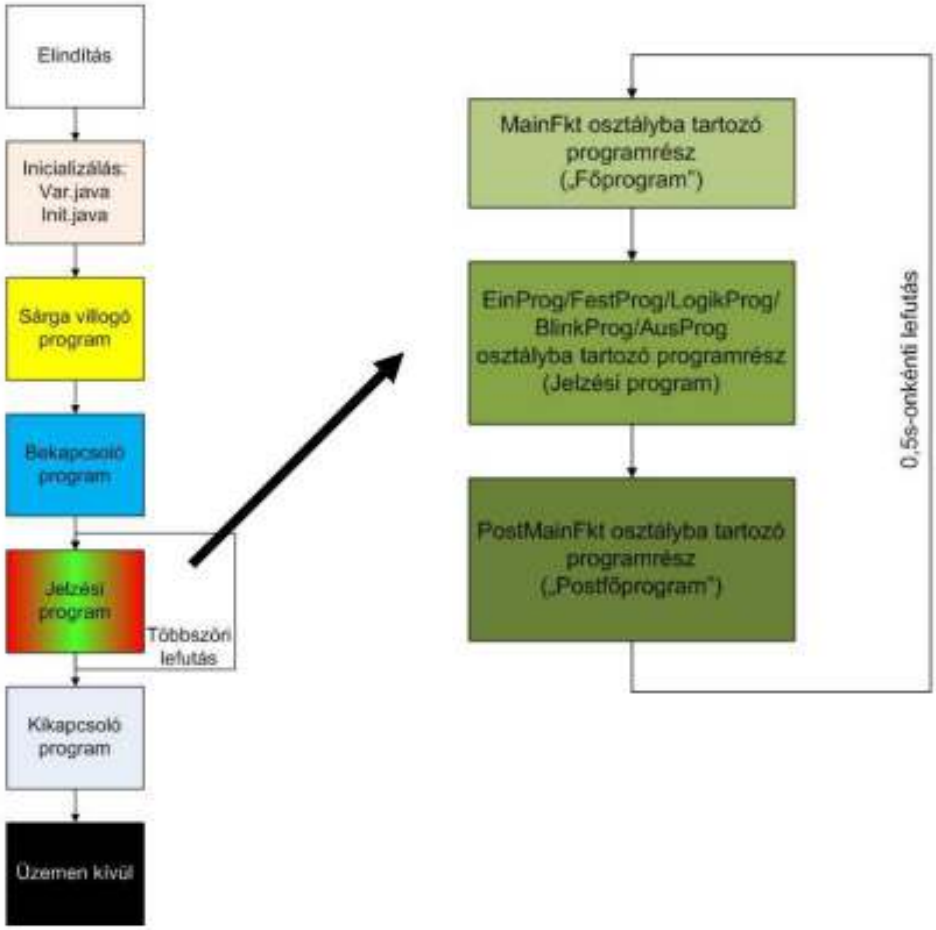
A készülék bekapcsolása után először a Var és Init osztályok futnak le. Ezt követően a rendszer előbb a sárga villogó programra (SvProg), majd a bekapcsoló programra (BeProg) áll át. Utóbbin keresztül jut el a berendezés az időtervnek megfelelő fixprogramhoz, vagy a forgalomfüggő programhoz (egyszóval a jelzési programhoz) [24].

A jelzési programban a berendezés belső, 0,5 másodperces ciklusideje alatt több fájl is lefut:

- Főprogram: itt vannak a minimális zöldidők megadva
- Jelzési program: jelzésterv, fázisok kapcsolása, forgalomfüggő logikák alkalmazása
- Postfőprogram: nyomógombok és detektorok alapállapotba állítása

Mikor a berendezés lekapcsolására kerül sor, akkor az aktuális jelzéstervből átáll a kikapcsoló programra (KiProg) és csak ezután kerül üzemen kívüli állapotra.

A teljes folyamat a 7. ábrán is jól végigkövethető.



9. ábra: A programrészek futási sorrendje és a jelzési program futása
(forrás: [24])

4.2.3.1. Var.java

A létrehozni kívánt objektumokat és változókat itt kell megnevezni, deklarálni. Mivel ezen osztály globális tartalommal bír, ezért ehhez, bármelyik más osztály hozzáférhet, tartalmát módosíthatják.

Új változó, illetve objektum létrehozásánál törekedni kell az egység megőrzésére, tehát az azonos osztályba tartozó objektumok változói egymás után következzenek, össze-

függő tömböt alkossanak, ezáltal a programkód átláthatóvá tehető, elkerülve a későbbi bonyodalmakat. Új elem hozzáadása a következőképpen történik:

- objektum deklarálása:

```
public static osztálynév objektumnév
```

- változó deklarálása:

```
public static változótípus változónév
```

A Var osztályban a következő objektumok kerülnek definícióra:

- Maga a forgalomirányító berendezés (Anlage)
- A berendezés által vezérelt csomópont(ok) (TeilKnoten)
- Csomópontok (Sg)
- Ismétlőjelzők (Wiederholer)
- Detektorok (Detektor), a gyalogos nyomógombok is idetartoznak.
- Be és kikapcsoló program (BeProg és KiProg)
- Sárgavillogó program (SvProg)
- Fixprogramok, illetve forgalomfüggő programok (FixProg, ForgfProg)
- Fénypontok típusa (LmpTyp)
- Jelzőcsoport típus (SgTyp)
- Egyéb objektumok.

A felsorolásban, a zárójelek között az objektumok eredeti nevei olvashatóak. Ezek legtöbbje német elnevezés, érthető, mivel a programot német cég készítette.

4.2.3.2. Init.java

A Var osztályban létrehozott változók kezdőértékkel történő ellátása, illetve az objektumok tényleges létrehozása ebben a részben fog megtörténni. A változók osztályában deklarált objektumok, az Init-ben konstrutorok segítségével jönnek ténylegesen létre, melyek legtöbbször kezdő paramétereket igényelnek. Összefoglalva ebben az osztályban inicializálást végzünk, innen az osztály elnevezése is.

Az egyes objektumfajták inicializálása külön metódusokban történik, így az osztály felépítése jól tagolt. Ezen metódusok nevéből (intialisiereX, ahol X az objektum típust jelöl) lehet következtetni a tartalmára.

A Var osztály elemein túl más osztálybeli objektumok létrehozása is itt történik:

- Közbensőidő mátrix (zwzMatrix)
- Napi és heti terv (TagesPlan és WochenPlan)
- Kapcsolókártyák és IO-kártya (SchalterKarte, IoKarte)
- Kapcsoló és IO csatornák (ShaltKanal, IoKanal)
- Egyéb

Objektumlétrehozás és inicializálás folyamata röviden:

```
Objektumnév = new osztálynév (paraméterlista)
```

Az ACTROS kódjának a „kulcsa”, a main metódus is itt foglal helyet. A berendezés elindulásakor ezt fogja a fordító legelőször meghívni, tehát a benne leírt inicializáló program kódok futnak le elsőnek.

4.2.3.3. FixProg.java és ForgfProg.java

A FixProg alosztályok a FestProg osztályba tartoznak. Ezek olyan a forgalomtechnikai programokat tartalmaznak, amelyek működésük során nem változnak: állandó ciklusidők és fix jelzőcsoport-vezérlés jellemzi ezeket, tehát nem forgalomfüggők. Két konstruktoruk van, melyek valamelyest eltérnek egymástól. A program megadása a K osztály BEKI metódusa segítségével történik, mely paraméterként a jelzőcsoport objektumát, valamint a szabad jelzés kezdetét (az előkészítő időt is hozzá kell számolni) és végét kell meg adni.

```
K.BEKI(Var.1.jelzőcsoport neve, kezdés, vég);
K.BEKI(Var.2.jelzőcsoport neve, kezdés, vég);
…
```

Ahhoz, hogy a rendszerünk forgalomfüggővé váljon, két dolgot lehet tenni. Az egyik, hogy több fix programot hozunk létre, melyek üzembeállását meghatározott időponthoz, vagy eseményhez kötjük. A másik lehetőség, hogy fix programok helyett forgalomfüggő programot (ForgfProg) alkalmazunk.

A LogikProg osztályba tartozó ForgfProg azáltal lesz forgalomfüggő, hogy képes bizonyos eseményekre (pl.: a telepített gyalogos nyomógombok megnyomása, meghatározott jármű szám elérése, stb.) megfelelően reagálni. Reagálás alatt, a működő fázis bizonyos részeinek megnyújtását, illetve átugrását értjük. Ezáltal lehet a jelzőcsoportokon keresztül, a felmerülő igényeknek megfelelő szabad jelzéseket és ciklusidőt biztosítani. A program megadása a FixProg-hoz hasonlóan történik, azzal a különbséggel, hogy ki kell egészíteni a forgalomfüggőséget leíró résszel, vagyis meg kell fogalmazni, hogy mely eseményekre, hogyan módosuljon a program.

5. A rendszer létrehozása

A központ és a terepi berendezés kapcsolata egy általános szerver-kliens kapcsolatként is értelmezhetjük: a szerver fogadja a kliensek kéréseit, majd ezekre megfelelő választ küld vissza.

Ez a fejezet a létrehozott szerver és kliens alkalmazások megvalósításának részletes leírását tartalmazza.

5.1. Kliens oldal

A rendszer megvalósításához, a már meglévő ACTROS forgalomtechnikai kódot kellett módosítani, kiegészíteni, valamint új programrészeket kellett hozzáadni. Az új funkciók, vagyis a detektor adatok átalakítása tovább küldésre a központnak, új, módosító paraméterek fogadása és implementálása, elkülönítve fognak futni az alap forgalomtechnikai részektől: külön, úgynevezett szálon fognak működni. Erre azért van szükség, hogy az alap, illetve az új kliens folyamatok ne akadályozzák egymást, azaz ne kelljen az egyiknek a másikra várnia. A szálak alkalmazásával a két rész futása nem soros módon történik, hanem kvázi-párhuzamosan vagyis mintha azok egyszerre, egy időben működnének. A szálakat tekinthetjük a program bizonyos részeinek, melyeket párhuzamosan is lehet futtatni. Ez úgy oldható meg, hogy a szálak mindegyike periodikusán kap bizonyos rendszeridőt. Ezzel növelhetjük a rendszer kihasználtságát, a művelet elvégzésének ideje rövidebb lesz, a soros feldolgozáshoz képest.

5.1.1. Módosítások, kiegészítések

Az újonnan bekerülő programrészek még önmagukban nem elegendőek a cél elvégzéséhez, a berendezés „alapprogramját” is módosítani kell. Fontosabb módosítások a következő osztályokban történtek:

- Var
- Init
- AltalanosResz

5.1.1.1. Var.java

A berendezés programozásához szükséges változók és objektumok, itt, ebben az osztályban kerülnek deklarációra. A rendszer működéséhez, az alábbi új változók bevezetésére volt szükség:

- *detszam* nevű egész szám (integer). Ez a változó a rendszerben lévő detektorok számát tárolja. Már itt értéket is kapott: 4, mivel a tanszéki berendezés négy detektorral rendelkezik.

```
public static int detszam=4;
```

- *jcsszam* nevű egész szám (integer). Ebben a változóban, az adott csomóponthoz tartozó jelzőcsoportok számát kell tárolni. Jelen esetben az érték 10 lesz, mivel itt, 4 db jármű, 4 db gyalogos, 2 db villogó jelzőcsoport van.

```
public static int jcsszam=10;
```

- *allj* nevű logikai változó (boolean). Ha működés közben valamilyen előre nem látható hiba történik (pl.: megszakad a kapcsolat a központi géppel), le kell állítani a kliens programot, pontosabban az azt működtető szálat. Kezdetei értékadás itt is volt: False (hamis), mert True (igaz) értéket akkor fog kapni, ha az előbb említett nem várt hiba bekövetkezik, tehát a program normális működése folyamán ez a logikai változó hamis.

```
public static boolean allj=false;
```

- *d* nevű, Detektor objektum tömb. A Detektor speciális objektum, nem a JAVA programnyelvhez tartozik, ez az ACTROS saját objektuma, mely a készülék detektorait reprezentálja. Tömbbe foglalásuknak, elsősorban kényelmi szerepük van: így nem kell a detektorokkal külön, egyesével foglalkozni, hivatkozni rájuk, adataikat lekérdezni, létrehozni, mert így már egységként, ciklusok segítségével tudjuk azokat kezelni. Ezáltal fokozódik a programkód tömörsége, egyszerűbb a kezelésük, viszont, ezáltal megnő a hibalehetőség is. A létrehozandó tömb elemeinek számát már meg kellett adni (detszam=4), így az objektumtömb a létrehozása a következőképpen jön létre:

```
public static Detektor[] d =new Detektor[detszam];
```

- *x* nevű egész szám (Integer) tömb. Ebben a tömbben lesznek tárolva az egyes detektorok által szolgáltatott jármű számok. A tömb létrehozása itt történik a tömb méretének (detszam=4) meghatározásával.

```
public static int[] x=new int[detszam];
```

- *zkv* nevű egész szám (Integer) mátrix. Ebben a tömbben lesznek tárolva a központ által, minden jelzőcsoporthoz kiszámított szabadjelzések kezdetei (beleszámítva az előkészítő időt) és végei. A tömb létrehozása szintén itt történik a kétdimenziós integer tömb méretének meghatározásával: A sorok száma *jcsszam*-mal lesz egyenlő, míg az oszlopok száma 2 lesz (1. oszlop a szabad jelzések kezdeteit míg, a 2. pedig a végeit tartalmazza), így egy meghatározott sor, az adott jelzőcsoporthoz tartozó értékeket foglalja magába.

```
public static int[][] zkv=new int[jcsszam][2];
```

- p nevű egész szám (integer), mely az éppen alkalmazott periódusidő (ciklusidő) értékét fogja tárolni.

```
public static int p
```

A rendszerben, ezeken túlmenően, hat új speciálisan, ehhez a rendszerhez kialakított forgalomfüggő programra van szükség, melyeket szintén deklarálni kell:

```
public static ForgfProgX prog5;
public static ForgfProgX prog6;
public static ForgfProgX prog7;
public static ForgfProgX prog8;
public static ForgfProgX prog9;
public static ForgfProgX prog10;
```

5.1.1.2. Init.java

A Var osztályban újonnan deklarált változók legtöbbjének van kezdeti értéke, viszont objektumoknak még nincs, így azokat az Init nevű osztályban inicializálni kell.

Ebben a fájlban az egyes elemek külön-külön helyen vannak inicializálva, például a detektorok az initialisiereDet részben, ahová az új négyelemű detektor tömbünk is került:

```
for(int i=0;i<Var.detszam;i++){
      Var.d[i]= new Detektor(Var.tk1,"Dt"+i,0,0,0,i+1);}
```

Az objektumtömb létrehozása a benne tárolandó elemek számának megadásán keresztül történik. A tömb elemeinek létrehozása úgynevezett konstruktorok segítségével történik. Ezek a konstruktorok már készen vannak, a forgalomtechnikát programozó személynek csak a megfelelő paramétereket kell megadni az objektum létrehozásához. A detektor objektum létrehozása esetén a paraméter lista a következő: melyik csomópont (meg kell adni, mert a készülék akár három csomópontot is tud kezelni), megnevezés, foglaltsági idő, maximális küszöb túllépés ciklusonként, nem foglaltsági idő és jelzőcsoport, ami jelen esetben el van hagyva.

Az új rendszerhez készített programok inicializálása az initialisiereProgs() metódusban történik:

```
Var.prog5=new ForgfProgX(Var.tk1, "P5", 5, 50, 3, 3, 0, 0);
Var.prog6=new ForgfProgX(Var.tk1, "P6", 6, 60, 3, 3, 0, 0);
Var.prog7=new ForgfProgX(Var.tk1, "P7", 7, 70, 3, 3, 0, 0);
Var.prog8=new ForgfProgX(Var.tk1, "P8", 8, 80, 3, 3, 0, 0);
Var.prog9=new ForgfProgX(Var.tk1, "P9", 9, 90, 3, 3, 0, 0);
Var.prog10=new ForgfProgX(Var.tk1, "P10", 10, 100, 3, 3, 0, 0);
```

Az itt alkalmazott konstruktor által megkövetelt paraméterek: csomópont, név, programszám, periódusidő (a 6 programban az eltérések csak itt vannak), zöldhullám pont „A”, zöldhullámpont „B”, várakozási idő, átállási idő;

A következő kiegészítés a hardver-inicializáló, vagyis az initialisiereHW részben történt. Ahhoz, hogy a berendezés megkapja a mérőkészülékek felől érkező adatokat, az ACTROS IO kártyáinak egyes csatornáit hozzá kell rendelni egy adott detektorhoz. Ez az IoKanal objektum segítségével történik. Jelen esetben, a négy detektorhoz, négy csatornát kell lefoglalni.

```
for(int i=0;i<Var.detszam;i++){
      new IoKanal(Var.d[i],io1,i+1);}
```

Az objektum létrehozásához szükséges paraméterek: detektor objektum azonosítója, IO kártya objektum azonosítója és a csatorna száma.

Az eredeti kód tartalmazott egy olyan utasítást, mely meghatározta, hogy melyik forgalomtechnikai program induljon el a berendezés elindítása után.

```
Var.tk1.setProgWunsch(Var.prog4, StgEbene.STG_VT_ANWENDER);
```

Erre a részre az új rendszerben nincs szükség, a kliens alkalmazás egyik osztályában dől el, hogy melyik programnak kell működni.

Az init osztályon keresztül indul el az egész forgalomtechnikai program, ugyanis itt található a main metódus, amiről azt kell tudni, hogy egy alkalmazás végrehajtásánál, a JAVA értelmező ezt a hívja meg elsőnek. A többi metódust, pedig a main fogja meghívni, tehát idekerül az új funkciók működtetéséért felelős szálobjektum, létrehozása és elindítása:

```
ismetelt ism=new ismetelt();
ism.start();
```

5.1.1.3. AltalanosResz.java

Az AltalanosResz osztályból a következő rész kell eltávolítani:

```
if (Var.derr.belegt())
      Var.tk1.setProgWunsch(Vt.KEIN_PROGRAMMWUNSCH,
      StgEbene.STG_VT_ANWENDER);
else
      Var.tk1.setProgWunsch(Var.prog4, StgEbene.STG_VT_ANWENDER);
```

Ez a rész mondja meg, hogy mi történjen, ha nem lép fel detektorhiba (ebben az esetben megadott program indítása) és mi történjen ennek ellenkezőjekor. A programok indítására, később, egy új osztály egyik metódusában kerül majd sor.

5.1.2. Új programrészek, új funkciók

Ez az alfejezet foglalkozik, azon új elemekkel, melyek eredetileg nem részei egy ACTROS alaprendszerének. Az új részek feladatuk szerint különböző osztályokba kerültek:

- Adatok formátumának átalakítása: Atalkit
- Adatok hálózaton keresztüli küldése és fogadása:AdatBeKi
- Az új funkciók összefogása: Ismetelt

5.1.2.1. Atalakit.java

Ez az osztály, illetve ennek a metódusai a továbbításra szánt adatok megfelelő alakra történő átalakítását, valamint a visszaérkező értékek, megfelelő változókhoz rendelését végzik. Átalakításra a hálózaton keresztül történő adattovábbításnál és fogadásnál van szükség, így megállapítható, hogy két metódusra lesz szükség.

Detadat metódus

A detektor értékek hálózaton keresztüli továbbítására több lehetséges mód is van, de jelen esetben az egyik legegyszerűbb módszer került alkalmazásra: A detektoradatok együtt, egy fix hosszúságú elemekből álló stringben kerülnek továbbküldésre. Az átalakítás a következőképpen történik: először a Var osztályban létrehozott *x* tömb elemeihez hozzárendeljük az azonos sorszámú detektorok által mért adatokkal, melyet a getVerkehrsStaerke függvénnyel tehetünk meg.

```
Var.x[i]=Var.d[i].getVerkehrsStaerke();
```

Ezt követően, az előbb feltöltött tömb elemeit szöveggé kell alakítani, ez lesz az *ertek* nevű string változó. Az *ertek* (program)ciklusonként felveszi az aktuális detektoradatok szöveggé konvertált értékét.

```
ertek=Integer.toString(Var.x[i]);
```

Ez még így nem megfelelő; ahhoz, hogy hozzáfűzzük az átküldendő *adat* nevű stringhez, mert abban, az egyes átalakított értékeknek állandó, jelen esetben három karakter hosszúságúnak kell lenni. Ahol az *ertek* egy karakter hosszúságú, vagyis ott a mért járműszám tíz alatt volt, annak az elejére be kell illeszteni két darab nullát. Ahol viszont tíznél több járművet mért a detektor, ott értelemszerűen egy darab nullával kell kibővíteni az *ertek* változót.

```
if (ertek.length()==1) {ertek="00"+ertek;}
else if (ertek.length()==2) {ertek="0"+ertek;}
```

Most már alkalmas alakra lett hozva a detektor érték, így már hozzá lehet fűzni az *adat*-hoz.

```
adat+=ertek;      //adat=adat+ertek
```

A fent leírt eljárásokat a DetAdat nevű metódus, függvény végzi el, melynek a visszatérési értéke (az utolsó detektor adat átalakításával és az átküldendő stringhez történő csatolás után) az *adat* lesz.

AdatZold metódus

A másik átalakítást végző függvény az AdatZold nevet viseli. Működését tekintve ellentéte az előbb leírt DetAdat nevű függvénynek. A központ felől érkező válasz, mely a kiszámított zöldidőket, valamint az azokhoz tartozó új ciklusidőt tartalmazza, hasonló formában, mint az DetAdat: fix hosszúságú részekből összeállított string, melynek a felépítése a következő:

Átküldött adat = 1. jelzőcsoport zöldidejének kezdete, 1. jelzőcsoport zöldidejének a vége, 2. jelzőcsoport, … , n. jelzőcsoport zöldidejének kezdete, n. jelzőcsoport zöldidejének a vége, a megállapított periódusidő.

Mivel az egyes részek (zöldidő kezdet, vég és ciklusidő), egyenként állandó, 3 karakter formájában vannak behelyezve az átküldött stringben, így azokat, az alábbi módon, könnyen vissza lehet alakítani:

```
for(int i=0;i<Var.jcsszam;++i){
  for(int j=0;j<2;++j){
    Var.zkv[i][j]=Integer.parseInt(sz.substring(i*6+j*3,i*6+j*3+3));}
```

A művelet zárásaként, az alkalmazandó periódus idő megállapítása következik, amit a kapott string utolsó 3 karakterének számadattá történő átalakításával kaphatunk meg.

```
Var.p=Integer.parseInt(sz.substring(sz.length()-3,sz.length()));
```

A szabadjelzés értékek apró módosításra szorulnak, mivel a központ a nem számítja bele az előkészítő időket. Így a szabadjelzések kezdetei előrébb tolódnak 2 másodperccel, figyelembe azt is, ha esetleg negatív szám jönne ki.

```
for(int i=0;i<Var.jcsszam;++i){
      if (Var.zkv[i][0]-2<0)
            Var.zkv[i][0]+=Var.p-2;
      else Var.zkv[i][0]-=2;
```

A detektorok lekérdezésének következő idejét is be kell állítani, a pontos adatok érdekében.

```
for(int i=0;i<Var.detszam;i++){
      Var.d[i].addVerkehrsStaerke(2*Var.p);
```

Ez esetben is ciklus segítségével történik a detektor-tömb elemeire a hivatkozás. A paraméternek megadott érték a berendezés azon belső ciklusainak a számát jelenti, amennyivel kell visszamenőleg megtekinteni a detektoradatokat. Az ACTROS belső ciklusának hossza 0,5 másodperc. Mivel az alkalmazott periódus időt a *Var.p* tárolja, ez idő alatt a belső ciklus 2·*Var.p*-szer futott le.

A periódusidő ismeretében a készülék már ki tudja választani a megfelelő (ciklusidejű) programot:

```
if (Var.p==50)
      Var.tk1.setProgWunsch(Var.prog5, StgEbene.STG_VT_ANWENDER);
else if (Var.p==60)
      Var.tk1.setProgWunsch(Var.prog6, StgEbene.STG_VT_ANWENDER);
else if (Var.p==70)
      Var.tk1.setProgWunsch(Var.prog7, StgEbene.STG_VT_ANWENDER);
else if (Var.p==80)
      Var.tk1.setProgWunsch(Var.prog8, StgEbene.STG_VT_ANWENDER);
else if (Var.p==90)
      Var.tk1.setProgWunsch(Var.prog9, StgEbene.STG_VT_ANWENDER);
else if (Var.p==100)
      Var.tk1.setProgWunsch(Var.prog10, StgEbene.STG_VT_ANWENDER);
```

Mint már korábban szó volt róla, jelen rendszerben, a központi számítógép által megállapított ciklusidők 50, 60, 70, 80, 90, illetve 100 másodperc értékeket vehetnek fel.

5.1.2.2. AdatBeKi.java

Ez az osztály felel az adatok hálózaton történő küldéséért és fogadásáért. Kezdésként, a központtal történő kapcsolat felvételéhez szükséges változókat, mint például a központ hálózati azonosítóját (*servercim*), valamint port számát (*serverport*), értékkel együtt meg kell adni:

```
public static String servercim="10.10.50.37";
public static int serverport=11111;
```

Majd deklarálni kell a szükséges objektumokat:

```
public static Socket csomag=null;
public static BufferedReader be=null;
public static PrintWriter ki=null;
```

Ahol:

- a Socket, a socket-alapú kommunikáció megvalósításához szükséges objektum, Egy kétirányú kapcsolat egyik oldalának vezérlését végzi.
- BufferedReader: adatfolyam olvasásához szükséges osztály. Az olvasás során az adatok pufferelődnek, így csökken a forrás adat eléréseinek száma.
- PrintWriter: Adatfolyamatok tartalmának kiíratására szolgálnak.

Az AdatBeKi osztály egy metódust tartalmaz: az ABK-t.

A metódus első feladata, az osztály elején deklarált objektumok létrehozása.

```
csomag=new Socket(servercim,serverport);
      be = new BufferedReader(
                  new InputStreamReader(csomag.getInputStream()));
ki=new PrintWriter(csomag.getOutputStream());
```

A *csomag* nevű, socket objektum létrehozásánál a paraméterként a központi számítógép, vagyis a szerver elérhetőségeit, vagyis a hálózati azonosítóját (*servercim*), valamint a portszámát (*serverport*), melyen keresztül fogadja a kliensek kéréseit. Ezt a socket-en keresztüli, be és kimenő adatfolyamokat kezelő objektumok (*be* és *ki*) létrehozása követi.

A kimenő adatfolyam tartalmát, vagyis a string formátumra alakított, lekérdezett detektor adatokat, a meghívott, Atalakit osztály DetAdat metódusa adja meg. A továbbítás végül a flush() metódus által történik meg.

```
ki.println(Atalakit.DetAdat());
ki.flush();
```

A járműszám értékek elküldése után, már érkezik is a válasz a központ felől, ami közetlenül az Atalkit osztály AdatZold metódusának van átadva.

```
Atalakit.AdatZold(be.readLine());
```

A feladatok elvégeztével, a socket-et és az adatfolyamokat le kell zárni a close metódus segítségével lehet megtenni.

```
csomag.close();
be.close();
ki.close();
```

Az itt leírt metódusban történő esetlegesen fellépő kommunikációs hiba (kivétel) során a Var osztályban definiált *allj* változó true, azaz igaz értéket kap, melynek hatására egy másik metódusban leállításra kerül a kliens program (és a berendezés fixprogramú vezérlésre áll át). A kivételek (a kivételes események) bekövetkezése során megszakad a program normál működése. Ezért ilyen eseményekre fel kell készíteni az alkalmazásunkat is, tehát kivételkezelést alkalmazunk. Azt a részt, ahol a hiba felbukkanása történhet egy úgynevezett try-blokkba kell helyezni. Ha a metódus eldob egy kivételt, akkor a futtató környezet megpróbálja elkapni, vagyis keresni egy kivételkezelőt.

```
throws IOException{
try{
…
catch(IOException
e){Var.allj=true;}
```

5.1.2.3. Ismetelt.java

Ez az osztály olyan Thread osztály leszármazottja. Az elvégzendő feladatok a run metódusba kerülnek. Az ismetelt osztály feladata, hogy amíg nem jelentkezik valamilyen hiba, vagyis amíg a Var.*allj* változó igaz értéket nem kap, addig az aktuális ciklus idővel megegyező időközönként meghívja az AdatBeki osztály ABK metódusát, ezáltal biztosítva a kliens programrész periodikus működését. A szál működésének szüneteltetésére a Thread.sleep metódust alkalmazzuk, melynek paraméterként meg kell adni, a szüneteltetés idejének hosszát, milliszekundumban, jelenesetben: Var.p ·1000 lesz (mivel a periódus idő másodpercben van megadva). Megjegyezendő, hogy az ABK metódus meghívása előtt is van egy szálszüneteltetés, melynek ideje 30 másodperc. Erre azért van szükség, mert a kliens program, az inicializáláskor azonnal elindul, így körülbelül 30 másodperces „előnnyel” fog rendelkezni mint a jelzési programmal szemben (mivel ez előtt, a bekapcsoló program is lefut), ami azért nem jó, mert csak néhány másodperces futási különbségre van szükség. Tehát, az adatcserének a központtal, pár másodperccel a programváltási pont előtt kell végbe mennie, így biztosítva, hogy az aktuális adatoknak megfelelő program, még időben kerüljön elindításra.

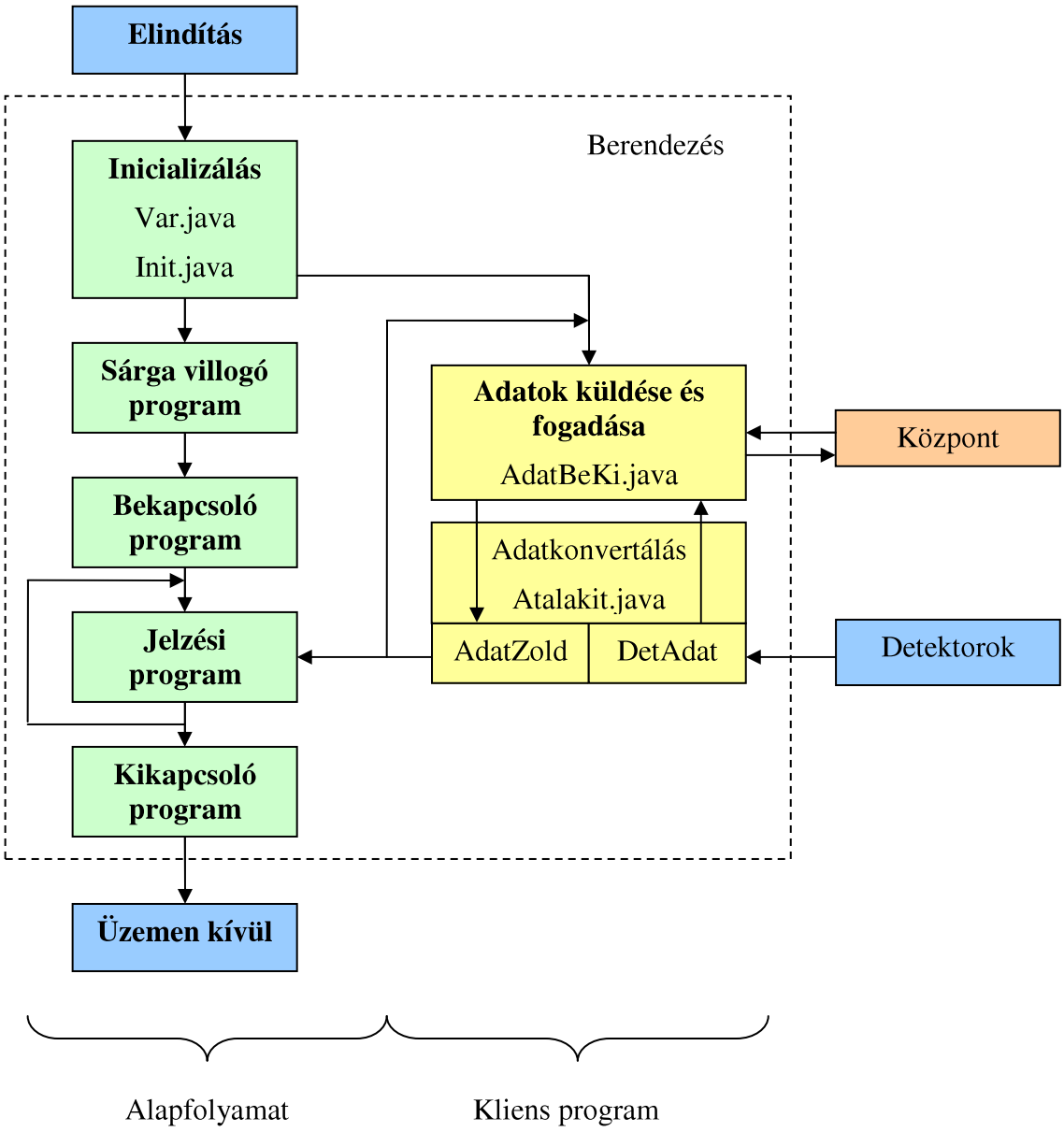
```
public class Ismetelt extends Thread
{
      public void run(){
            while (Var.allj!=true){
                  try {
                        Thread.sleep(30000);
                  } catch (InterruptedException e1) {
                        e1.printStackTrace();
                  }
                  try {
                        AdatBeKi.ABK();
                        try {
                              Thread.sleep(Var.p*1000);
                        } catch (InterruptedException e) {
                              e.printStackTrace();
                        }
                  }catch (IOException e) {
                        e.printStackTrace();
            }
}
```

5.1.2.4. ForgfProgX.java

A jelen megvalósításnál felhasznált forgalomfüggő programok osztálya. Felépítése hasonlít a többi forgalomfüggő program osztályéra, az eltérés a szabadjelzések időpontjainak megadásában van. Itt is a K osztály BEKI metódusa van alkalmazva, melynek paraméterlistájába az adott jelzőcsoport objektum, és a megfelelő szabad jelzés kezdeti és vég időpontjai kerülnek. Az utóbbi kettő megadása, a szerver felől érkezett, időponti értékeket tartalmazó *zkv* tömbből történik. Emlékeztetőül, a zkv mátrix első (pontosabban a 0.) oszlopa a kezdeti időket, a második (pontosabban az 1.) oszlopa pedig a befejezési időket tartalmazza.

```
K.BEKI(Var.j11, Var.zkv[0][0], Var.zkv[0][1]);
K.BEKI(Var.j21, Var.zkv[1][0], Var.zkv[1][1]);
K.BEKI(Var.j31, Var.zkv[2][0], Var.zkv[2][1]);
K.BEKI(Var.j41, Var.zkv[3][0], Var.zkv[3][1]);
K.BEKI(Var.gy51, Var.zkv[4][0], Var.zkv[4][1]);
K.BEKI(Var.gy61, Var.zkv[5][0], Var.zkv[5][1]);
K.BEKI(Var.gy71, Var.zkv[6][0], Var.zkv[6][1]);
K.BEKI(Var.gy81, Var.zkv[7][0], Var.zkv[7][1]);
K.BEKI(Var.sv91, Var.zkv[8][0], Var.zkv[8][1]);
K.BEKI(Var.sv101, Var.zkv[9][0], Var.zkv[9][1]);
```

5.1.3. Kliens alkalmazás működésének összefoglalása



10. ábra: a berendezés módosított működésének sematikus ábrája

A forgalomirányító berendezés elindítását követően, előbb betöltésre, majd inicializálásra kerülnek az egyes változók és objektumok. Ezzel együtt, az init osztályban elindításra kerül a kliens alkalmazás is, amely mint már szó volt róla, külön szálon, az alapfolyamatokkal párhuzamosan fog ciklikusan, újra és újra lefutni. A szál működése az ismetelt nevű osztályban került leírásra, melynek működése egyszerű: meghívja az AdatBeKi osztály ABK metódusát, majd az éppen aktuális periódusidőnek megfelelően újraindítja önmagát, feltéve, ha nem jelentkezik valamilyen hiba, mert ilyenkor leáll a teljes kliens alkalmazás. Az ABK metódus felel a berendezés és a központi számítógép

közötti kapcsolat kezeléséért, így az adatok küldéséért és fogadásáért is. Küldéskor a meghívják az Atalkit osztály DetAdat metódusát, mely visszatérési értékként, a lekérdezett detektoradatok megfelelő formátumra hozott alakjaival szolgál. Ez az átalakított érték együttes lesz továbbítva a szerver felé. Az elküldött adatoknak megfelelő válasz, a központ felől, szinte azonnal érkezik is vissza, azonban ez is átalakításra szorul: az ABK metódus meghívja az Atalakit osztály, AdatZold metódusát átadva annak a válasz adatot. Az AdatZold a válasz üzenetet átkonvertálja a forgalomtechnikai program számára értelmezhető formára. A kapott adatok alapján kiválasztásra kerül a megfelelő ciklusidejű program, amely a következő átadási pontnál életbe lép.

A teljes folyamat összefoglaló a 10. ábrán is végigkövethető.

5.2. Szerver oldal

A szerver, vagyis a központi szoftver két fő részre bontható:

- Egy alapfolyamatra, mely a felhasználók számára láthatatlan. A háttérben működik, kiszolgálja a klienseket: fogadja a kérésüket, számításokat végez, majd a választ továbbítja, vissza a kliensnek, ezek mellett archiválási teendőket is elvégzi.
- Egy felhasználói interfészre, amelyen keresztül a felhasználó meg tudja változtatni a szabályozáshoz szükséges, az adott csomópont jelzőire vonatkozó, megengedett legkisebb, valamint a legnagyobb zöldidőket. Emellett különböző információkat is szolgáltat:
    - Közbenső idő mátrix
    - Csomóponti felépítés (ábra)
    - Általános adatok (megnevezés, IP cím, periódus idő)
    - Archivált adatok előhívása, ábrázolása
    - Aktuális adatok alapján történő fázisterv megrajzolása

A központ, az általa ismert berendezésekről és az azok által vezérelt csomópontokról tartalmazó információkat, egy adatbázisból olvassa ki. Az adatbázis egy külön állományban van eltárolva (csp.mat). Az adatbázisnak a következő mezői vannak:

Mezőnév	Típus	Megjegyzés
nev	Szöveg	A forgalomirányító berendezés megnevezése
ipcim	Szöveg	A forgalomirányító berendezés hálózati azonosító címe
kozb	Mátrix	Az adott csomópontra vonatkozó közbensőidő mátrix
tipus	Vektor	Az adott csomópontban szereplő jelzőcsoportok típusa. (1: jármű, 2: kiegészítő, 3: kerékpár, 4: villamos, 5: gyalogos, 6: villogó)

fazis	Vektor	Az adott csomópont jelzőcsoportjai, mely fázisba vannak sorolva.
det	Vektor	Az adott csomópontban szereplő detektorok, mely jelzőcsoporthoz tartoznak
villogo	Mátrix	Az adott csomópont gyalogos-átkelőhelyéhez tartozó villogó
offset	Vektor	A hálózat más elemeivel történő összehangoláshoz szükséges időbeli eltolás mértéke
kep	Szöveg	Az adott csomópont helyszínrajzának elérhetőségi útvonala

1. Táblázat: A statikus adatokat tartalmazó adatbázis elemei

A nem-állandó, a számítások során, folyamatosan változó adatok, valamint az adott csomópont vezérléséhez szükséges zöldidő korlátok is tárolásra kerülnek. Egy külön adatbázisban, az Adatok.mat-ban kapnak helyet az ideiglenes változók. Az adatbázis a következő mezőkkel rendelkezik:

Mezőnév	**Típus**	**Megjegyzés**
zkv	Mátrix	A jelzőcsoportok szabad jelzéseinek kezdetét és végét tartalmazza
p	Szám	Az adott csomópontra kiszámított ciklusidő
zik	Vektor	Az adott fázis kiemelt irányához tartozó zöldidő korlát

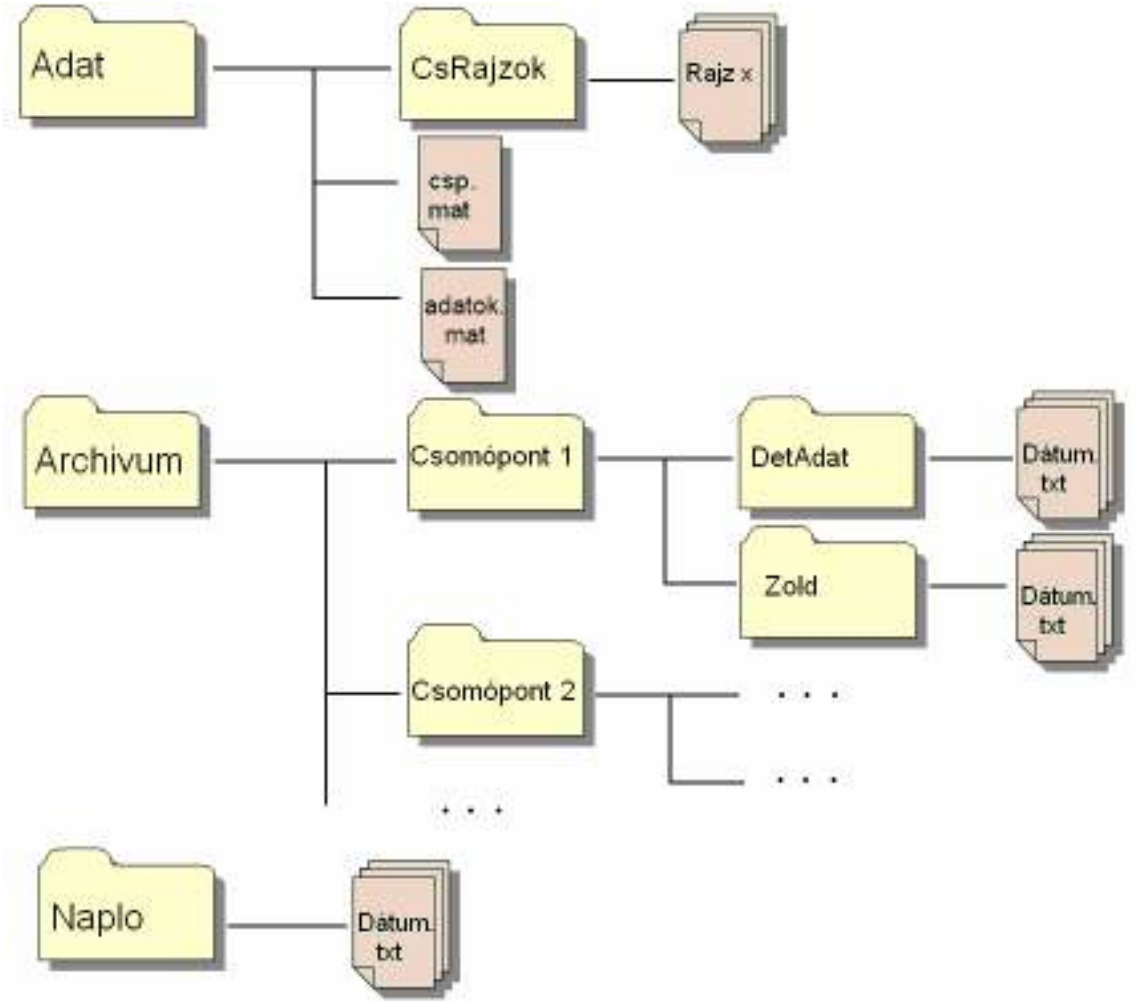
2. Táblázat: A dinamikus adatokat tartalmazó adatbázis elemei

A szerver egyik legfontosabb feladata a mért és számított adatok archiválása, illetve az események naplózása, a jövőbeli forgalomtechnikai számítások, az adatok ábrázolása, a működés ellenőrzése, illetve a további fejlesztések céljából. Ennek megfelelően lényeges, hogy az adatokat áttekinthető formában tárolja a rendszer, így megfelelő könyvtárstruktúrára van szükség.

A szerver program működése során a beérkező kéréseket naplózza, egy közös fájlba elmenti a kérést intéző kliens IP címét, valamint a rendszeridőt. Minden nap új naplófájl kerül szerkesztésre, így az egyszerűség kedvéért az adott fájl neve, az aktuális dátum lesz. A naplófájlok a szerver munkamappájában található „Naplo” nevű mappában kerülnek elmentésre.

A lementett beérkező és számított adatok egyaránt az „Archivum” nevű mappába kerülnek, megfelelően tagolódva: először csomópont neve szerint, másodszor pedig aszerint, hogy a kliens felől érkezett adatról, vagy a központi számítógépben számított adatról van-e szó. Előbbi esetben a „DetAdat” nevű mappába, míg utóbbi esetben a „Zold” nevű mappába kerülnek archivált fájlok, melyek elnevezése az előző esethez hasonlóan az aktuális dátum lesz.

Végül meg kell említeni, az archiválástól független, a számításokhoz, adatábrázolásokhoz nélkülözhetetlen elemek könyvtárát az „Adat”-ot. Itt található meg a csomópontok helyszínrajzait tartalmazó „CsRajzok” mappa, valamint, a központi gép által kezelt csomópontokra vonatkozó állandó adatokat tartalmazó állomány (csp.mat) és a számítások során létrejött, állandóan változó forgalomtechnikai paramétereket és a zöldidő korlátokat tartalmazó adatok.mat is. A teljes könyvtár és fájl szerkezet a 11. ábrán látható.



11. ábra: A központi szoftver által létrehozott könyvtár szerkezet

5.2.1. Az alapfolyamat részei és azok működése

A központi szoftver háttér feladatait ellátó rész, több egységre lett osztva, aszerint, hogy milyen részfeladatot kell ellátniuk:

- Fő eljárás, amely a többi részfunkció meghívásáért felel, valamint a hálózaton keresztül történő adat fogadásért és küldésért. (server.m)
- A berendezés felől érkező detektor adatok archiválása (mentes.m)
- A kliens gépektől érkező (AdatX.m), valamint a klienseknek szánt (ZAdat.m) adatok megfelelő formára hozása.

- A szerver, számításokat elvégző függvényei, eljárásai:
  - Periódusidő kiszámítása (peridoszam.m)
  - Fázisok zöldidőinek kiszámítása (teloszt.m)
  - Fázisterv megállapítása (ptervszam.m)

5.2.1.1. Server.m

A szerver elindulásakor a legelső lépés az irányításhoz szükséges adatok beolvasása. A mat kiterjesztésű állományok, melyek az összes, a központ által ismert csomóponti statikus és dinamikus adatokat tartalmazzák, a load nevű eljárással kerülnek a központ rendszerébe, úgy, hogy egy megadott változóhoz csatoljuk azokat.

```
load('csp.mat','csp');
load('adatok.mat','adatok');
```

Mivel a kommunikáció a kliensekkel, hálózaton keresztül történik, ezért itt is létre kell, hozni a kapcsolat létrehozásához és az adatküldések és fogadások elvégzéséhez szükséges objektumokat:

```
scsomag=ServerSocket(11111);
csomag=scsomag.accept;
be=BufferedReader(InputStreamReader(csomag.getInputStream));
ki=PrintWriter(csomag.getOutputStream,true);
```

- Az *scsomag* Serversocket objektum. Paraméterként az adott gép, azon port számát kell megadni, melyen keresztül a kliensek kérését fogja várni.
- A kérések fogadásánál egy új socket jön létre: a *csomag*, melyen keresztül fog végbe menni a kommunikáció a klienssel.
- *Be*: BufferedReader: adatfolyamok olvasását szolgáló objektum
- *Ki*: PrintWriter: adatfolyamok írását végzi.

Ezek az objektumok JAVA objektumok, nem képzik a MATLAB szerves részét. Működésükhöz, importálni kell a megfelelő JAVA csomagokat:

```
import java.net.*
import java.io.*
import java.lang.*
```

Ahhoz, hogy az adott kliens kérésére megfelelő választ tudjon adni a központ, előbb azonosítani kell azt. A legegyszerűbb módszer erre, ha a beérkező adatcsomagból lekérdezzük a feladó hálózati azonosítóját (*cim*), mivel ez állandó, nem változik, így egyértelműen azonosítja a készüléket. A címet a betöltött adatbázison végig kell futtatni, egyezés esetén, a későbbi számításokhoz elengedhetetlen *aktualis* nevű változó megkapja az aktuális berendezés, az adatbázisban szereplő egyedi sorszámát.

```
cim=char(csomag.getInetAddress());
cim=cim(2:length(cim));
aktualis=0;
for i=1:length(csp)
  if csp(i).ipcim==cim;
     aktualis=i;
     break;
  end;
end;
```

A beolvasott adatsor(*keres*) átkonvertálására az AdatX függvény lett alkalmazva, amely a *detektor_adatok* nevű vektorral tér vissza, amely nevéből adódóan a detektor adatok számértékeit tartalmazza.

```
keres=readLine(be);
detektor_adatok=AdatX(keres);
```

Most már készen áll a rendszer a számítások elvégzéséhez. Először a periódusidő megkerül megállapításra, a peridoszam függvény segítségével, mely paraméterként az éppen aktuális ciklus időt (*P*), és a detektor adatokat (*detektor_adatok*) követeli meg.

```
peridoszam(P,detektor_adatok);
```

Ezt követően a fázisok zöldidő kerülnek kiszámításra. Erre a feladatra a teloszt függvény készült, melynek használatához meg kell adni az összes szabad zöldidőt (*tz*) ami megegyezik a periódus idő (*p*) és a fázisok közötti közbenső idők (sum(*szunet*)) különbségével, a zöldidő korlátokat tartalmazó mátrixot (*zik*), valamint detektor adatokat tartalmazó tömböt (*detektor_adatok*). A zöldidő korlátok alkalmazása előtt, azok egy kisebb átalakításon mennek keresztül, amire azért van szükség, mert a rendszer a korlátozó értékek 60 másodperces ciklusidőre vannak megállapítva, így ennek megfelelően arányosan lesz megváltoztatva, de csak a felső határ, az alsó nem változik.

```
zik(:,2)=round(zik(:,2)*P/60);
tz=P-sum(szunet)
...
fazis_z=teloszt(tz,zik,detektor_adatok);
```

A válasz üzenet (*valasz*) előállítása a ZAdat függvény segítségével történik, melynek paraméterlistájába meg kell adni a fázistervet tartalmazó mátrixot és az aktuális periódusidő értékét. A továbbításra a println metódus lett meghívva.

```
valasz=ZAdat(zkv,P);
ki.println(valasz);
```

A szerver szolgáltatatásai közé tartozik a kapott és kiszámított adatok elmentése is. Az adatok rögzítését, a mentésre szánt adatok típusa alapján, három fázisra bonthatjuk:

- Az adott terepi gép felől érkezett adatok mentése
- A kiszámított szabadjelzések adatainak mentése
- A kérések megjelenésének külön regisztrálása

A mentés, a következő alpontban tisztázott mentes nevű függvény segítségével történik. A szóban forgó függvény három paraméter kér:

- A cél mappa nevét (*mappa1 / mappa2 / mappa3*)
- A cél fájl nevét (*file*)
- A lementendő adatot (*keres / cím / valasz*)

```
mappa1=['Archivum\' csp(aktualis).nev '\DetAdat\'];
mappa2=['Archivum\',csp(aktualis).nev,'\Zold\'];
mappa3='Naplo\';
file=[datestr(now,'yyyy-mm-dd'),'.txt'];
mentes(mappa1,file,keres);
mentes(mappa3,file,cim);
mentes(mappa2,file,valasz);
```

A központi műveletet a kapcsolatbontás és az adatfolyamok lezárása zárja.

```
csomag.close;
scsomag.close;
be.close;
ki.close;
```

5.2.1.2. Mentes.m

Egy forgalomirányító szoftvertől elvárható a kliensek detektor adatainak, számított értékeinek és általános események archiválása, a későbbi felhasználás érdekében. Ennek a folyamatnak a központi gépen kell végbemennie és nem a helyi berendezésben, mert az utóbbinak szűkös memória erőforrásai vannak, illetve így sokkal áttekinthetőbb és egyszerűbb az irányítás, ha minden adat egy helyen van. A létrehozott, speciális archiváló eljárás három bemeneti paramétert követel meg: a célmappa és a célfájl nevét, valamint az elmenteni kívánt adatot (*targy*).

```
function mentes(mappa,file,targy)
```

Meghívásakor, először leellenőrzi a célmappa meglétét. Ha nem létezik ilyen nevű, akkor létrehozza azt.

```
    if (exist(mappa)==0);
        mkdir(mappa);
    end;
```

Ezután létrehozunk, a JAVA IO csomagjának importálásával, egy adatfolyamat tartalmának kiírását szolgáló *fajlba* nevű, PrintStream objektumot. Ezt csatolnunk kell valamilyen adatfolyamhoz, jelen esetben a megadott célfájlba író fájlfolyamhoz (FileOutputStream). Az utóbbi második paramétere true, ami azt jelenti, hogy, ha létezik az első paraméterben megadott célfájl, akkor, nem írja felül azt, hanem az új adatot hozzá fűzi. Miután létrejött az adatfolyam objektum, már lehet is használni az adatok fájlba írására, amihez a print, illetve a println metódusokat alkalmazzuk.

Először a rendszeridő kerül kiírásra. Ehhez a datestr függvény alkalmazzuk, ami a paraméterlistájában megadott dátumot (jelen esetben a rendszeridőt, amit a now függvénynyel kérdezünk le), a szintén paraméterként megadott formátumban állít elő.

Ezt követően kerül a *targy* változó értéke is írásra. Az előbb kiírt két szöveg egy sorba kerül, közöttük tetszőlegesen megállapított '---' karaktersorozat áll.

```
    import java.io.*
    …
    fajlba=PrintStream(FileOutputStream([mappa file],true));
    fajlba.print([datestr(now,'HH:MM:SS') '---']);
    fajlba.println(targy);
    fajlba.close;
```

5.2.1.3. AdatX.m

Meghívása paraméteresen történik. A paraméter, a kliens felől érkező, a mért járműszámokat tartalmazó string (*sz*), míg a visszatérési értéke pedig az ezeket, az adatokat tartalmazó mátrix lesz (*x*). Az átalakítási művelet egyszerű, mivel a detektor adatok állandó karakterszámmal (3 darab) vannak jelen az átküldött stringen belül, tehát ismerjük azok elhelyezkedését. Ennek tudatában, a substring függvény segítségével ki lehet ezeket emelni a teljes szövegből, majd a str2num függvény segítségével numerikus adattá alakítani azokat.

```
function x = AdatX(sz)
    for i=0:(length(sz)/3-1)
        x(i+1)=str2num(substring(sz, i*3,i*3+3));
    end
end
```

5.2.1.4. A számításokat elvégző függvények

Három számítást végző függvény van, melyek algoritmusai korábban már leírásra kerültek. Az számításokat végző függvények:

- Periódusidő kiszámítása (peridoszam.m): 4.1.3. alfejezet
- Fázisok zöldidőinek kiszámítása (teloszt.m): 4.1.4.alfejezet
- Fázisterv megállapítása (ptervszam.m): 4.1.5. alfejezet

5.2.1.5. ZAdat.m

A ZAdat függvény működése ellentétes az AdatX függvénnyel. Hasonlóan a kliens Atalakit osztály DetAdat metódusához, itt is fix hosszúságú stringbe kell foglalni az elküldendő adatot, ami jelen esetben a kiszámított szabad jelzések kezdő és befejező időpontjai lesznek. Az átalakítási módszer során a num2str függvénnyel szöveg típusúvá (*ertek*) konvertált adat elejét, karakterszámtól függően kell kiegészíteni nullákkal, ahhoz, hogy az állandó 3 karakterszám meglegyen. Majd ezt, hozzá kell adni, az strcat nevű függvénnyel, ahhoz a stringhez (*adat*), ami végül a függvény visszatérési értéke lesz, amihez még utoljára az aktuális ciklusidő, hasonlóan megformázott alakja is hozzáadódik.

Az egyes szöveggé konvertált számadatok három karakter hosszra pótlását egy úgynevezett „subfunction”, egy belső függvény (kiegeszit) végzi, mely paraméterként a kipótolandó stringet kéri.

```
function adat = ZAdat(zkv,P)
    function uj=kiegeszit(regi)
        switch (length(regi))
            case (1)
                regi=['00' regi];
            case (2)
                regi=['0' regi];
        end
        uj=regi;
    end

  adat='';
  for i=1:length(zkv(:,1))
      for j=1:2
        ertek=num2str(zkv(i,j));
        ertek=kiegeszit(ertek);
        adat=[adat ertek];
      end
  end

  adat=[adat kiegeszit(num2str(P))];

end
```

5.2.1.6. Cspfel.m

Ez a programrész az új csomópontok felvételére és a meglévők szerkesztésére szolgál, tehát az új rendszer csomóponti adatbázisának a kezelője. Benne, az 1. táblázat alapján, az adatok, egy struktúrákból álló tömbben vannak elhelyezve, a következőképp:

```
csp(1)=struct('nev','Teszt csomópont','ipcim','192.168.1.1'…
csp(2)=struct('nev','Csömöri út - János utca',…
…
```

A program végén a teljes adathalmaz mentésre kerül, egy speciális, MAT kiterjesztésű MATLAB fájlba, amiről annyit kell tudni, hogy egy „munkaterület” változóit lehet benne tárolni.

A mentés során a save parancsot alkalmazzuk, melynek paraméter listájába meg kell adni a cél fájl nevét (csp.mat), valamint az elmentésre váró változókat (jelen esetben a csp-t, vagyis a struktúra-tömböt).

```
save('csp.mat','csp');
```

5.2.2. A felhasználói interfész

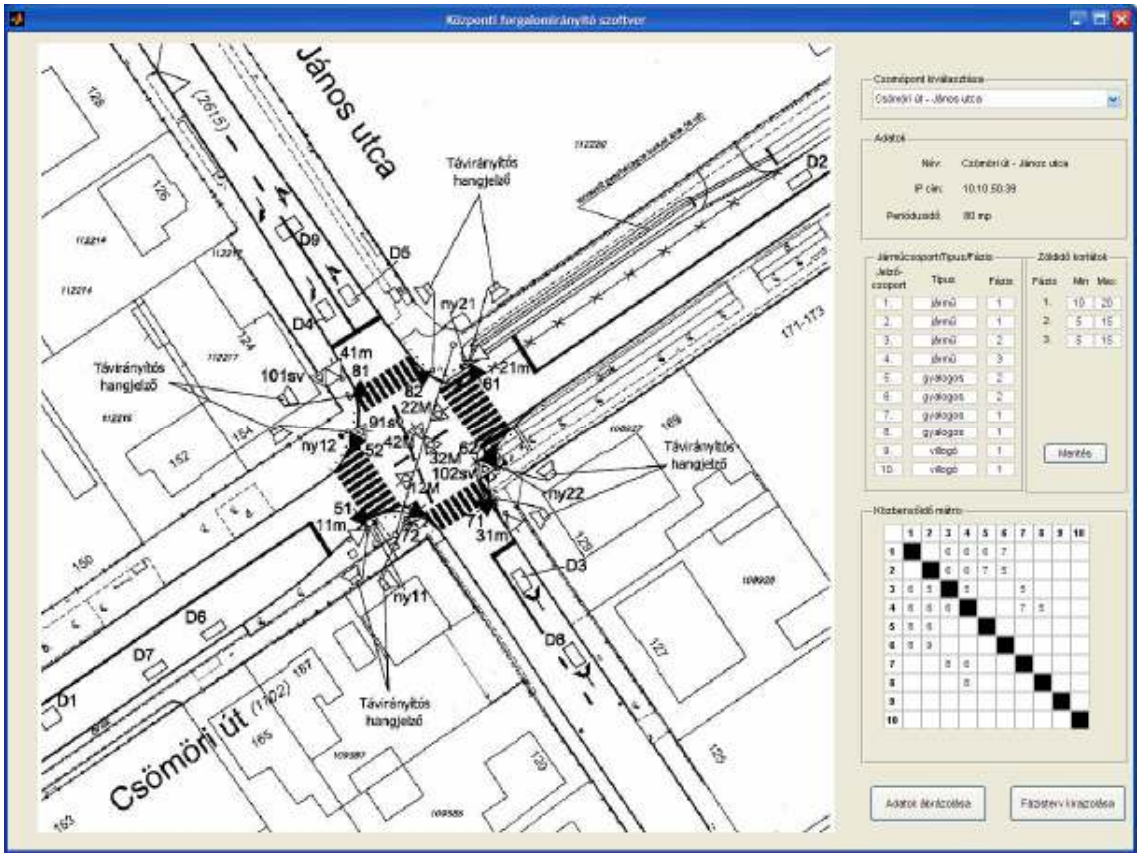
Az irányítási feladat elvégzésénél előnyt jelent, ha a van egy olyan kezelői felület, amelyen a keresztül a felhasználó könnyen hozzáfér az irányítás során született adatokhoz, sőt, akár a forgalomtechnikai paraméterek módosításával be is tud avatkozni. Az ehhez hasonló kezelői felületektől elvárható jellemzők:

- Grafikus, vizuális elemek alkalmazása
- Képes az adatok valamilyen formában történő megjelenítésére
- Felhasználóbarát
- Áttekinthető
- Könnyen kezelhető

5.2.2.1. Főablak

A Fo.m és a fo.fig fájlok tartalmazzák a szerver alkalmazás főablakának vizuális elemeit és működésének leírását. A 12. ábrán is látható a főablak felépítése. A megjelenített felület nagy részét a kiválasztott csomópont helyszínrajzának megjelenítésére alkalmas grafikus elem teszi ki. Természetesen a képet csak akkor tudja megjeleníteni, ha a csomóponti adatokat tároló adatbázisban, a megadott csomóponthoz meg van adva egy létező képnek az elérési útvonala. Az ablak jobb oldalán található elemei fentről, lefelé:

- Legördülő lista, mely a megtekinteni kívánt csomópont kiválasztására szolgál.
- Általános információk: a csomópont megnevezése, a készülék IP címe, az aktuálisan működő periódus idő
- További információk a csomópontról, táblázatos formában: jelzőcsoport sorszáma, annak típusa (jármű, villamos, gyalogos…) és a fázisának sorszáma
- A forgalomszabályozást korlátozó adatok megadására szolgáló beviteli mezők és az azok elmentését végző gomb.
- Az aktuális csomóponthoz tartozó közbensőidő mátrix
- Két gomb, melyekkel, a kapott és számított adatok grafikus megjelenítését végző eljárásokat lehet meghívni. Az egyik az aktuális fázisterv kirajzolását, míg a másik, a járműszámok, illetve a számított szabadjelzések ábrázolását végzi.



12. ábra: A központi szoftver indítóablaka

5.2.2.2. Adatábrázolás

A főablakon keresztül további adatokat kérhetünk le a kiválasztott csomópontról. Ilyen például a kapott járműszámok, valamint a kiszámított szabadjelzések lehívása, megadott időpontra vonatkozóan. Ilyenkor célszerű a nagy mennyiségű adathalmazt, az ember számára könnyebben értelmezhető formára hozni, ennek egyik legjobb módszere a grafikus megjelenítés. Jelen program is tartalmaz ilyen funkciót. A számításokat az m kiterjesztésű fájlban, míg az ablak vizuális megjelenítéséért a fig kiterjesztésű fájl felel.

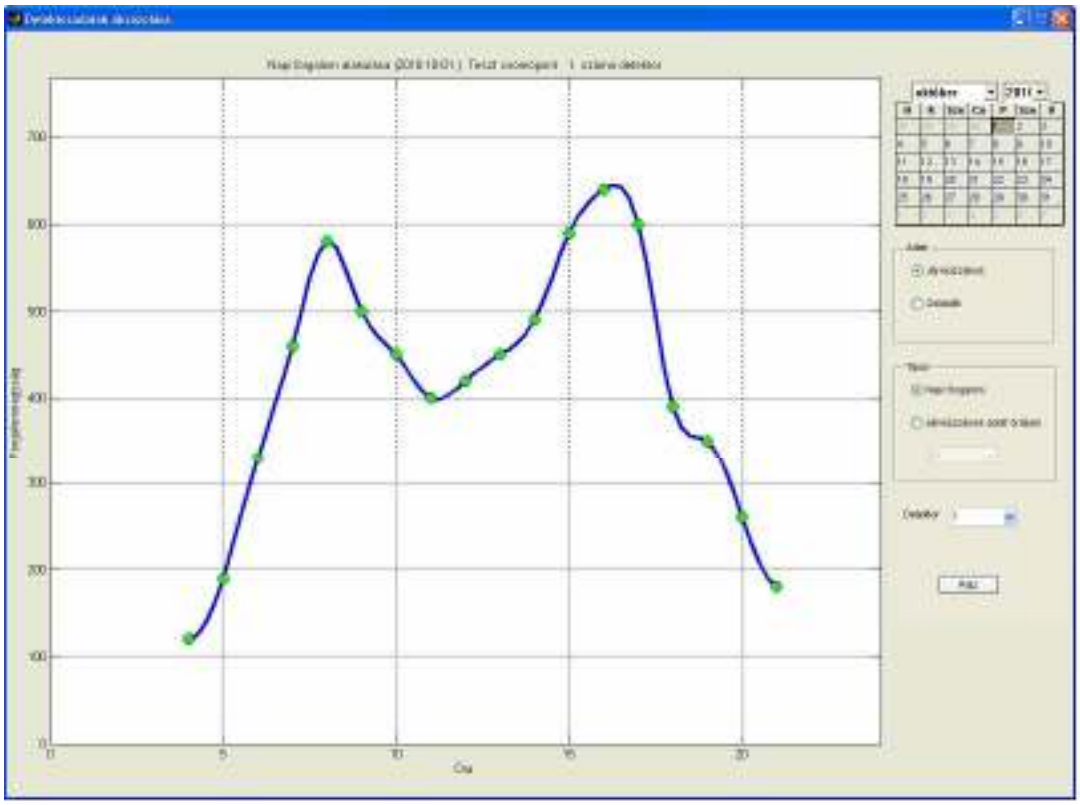
Ha a felhasználó használni akarja ezt a funkció, akkor korlátozó paraméterek kell megadni, a kívánt eredmény elérése érdekében. Ez már a főablaknál elkezdődik, ugyanis ott kell kiválasztani, hogy mely csomóponttal kívánunk foglalkozni. A további paramétermegadás már ennél az ablaknál folytatódik:

- A vizsgált nap dátumának megadása a naptár vezérlőelemen (ennek a hivatalos neve: Microsoft Naptár Vezérlőelem 11.0) keresztül történik.
- Ábrázolandó adat megadása: járműszámok vagy zöldidők
- Detektor kiválasztása egy legördülő-listából, ami az adott csomóponthoz tartozó detektorok sorszámait tartalmazza
- Adatok ábrázolásának módja. Az igények kielégítése végett az adatok ábrázolása kétféle képen történhet: választható a járműszámok közvetlen megjelenítése,

megadott órára vonatkozóan, illetve választható az adatok órákba foglalásával, a napi járműforgalom ábrázolása is. Ennek megadása a megfelelő választógomb (Radiobutton) megnyomásával történik.

- Ábrázolási módtól függően, óra megadása

Ezek alapján az ablak felépítése a következő ábrán (13. ábra) látható módon történt:



13. ábra: A detektoradatok ábrázolása

A paraméterek elfogadása és a diagram előállítása a „Rajz” feliratú gomb megnyomásával megy végbe. Mint már korábban szó volt arról, hogy az adatok a mérés dátuma szerint vannak elnevezve, így ez alapján a rendszer ki tudja választani a megfelelő fájlt. Abban az esetben, ha a kiválasztott időpontban nem létezik a detektoradatok archívuma, azt a rendszer „Nincs Adat!” hibaüzenettel jelzi.

Az elkészült diagramok vízszintes tengelyén az idő, függőleges tengelyén járműszámok, illetve zöldidők vannak reprezentálva.

Az adatok adott órára vonatkozó lekérdezése estén, oszlopdiagramot kapunk, melyen az egyes oszlopok magassága az adott időponthoz tartozó járműszámmal, vagy a szabadjelzés tartamával lesz arányos.

Teljes napra vonatkozó lekérdezésnél, az archívumból betöltött adatok, órás csoportokban, a forgalomnagyság ábrázolásánál összeadva, a szabadjelzéseknél átlagolva fognak

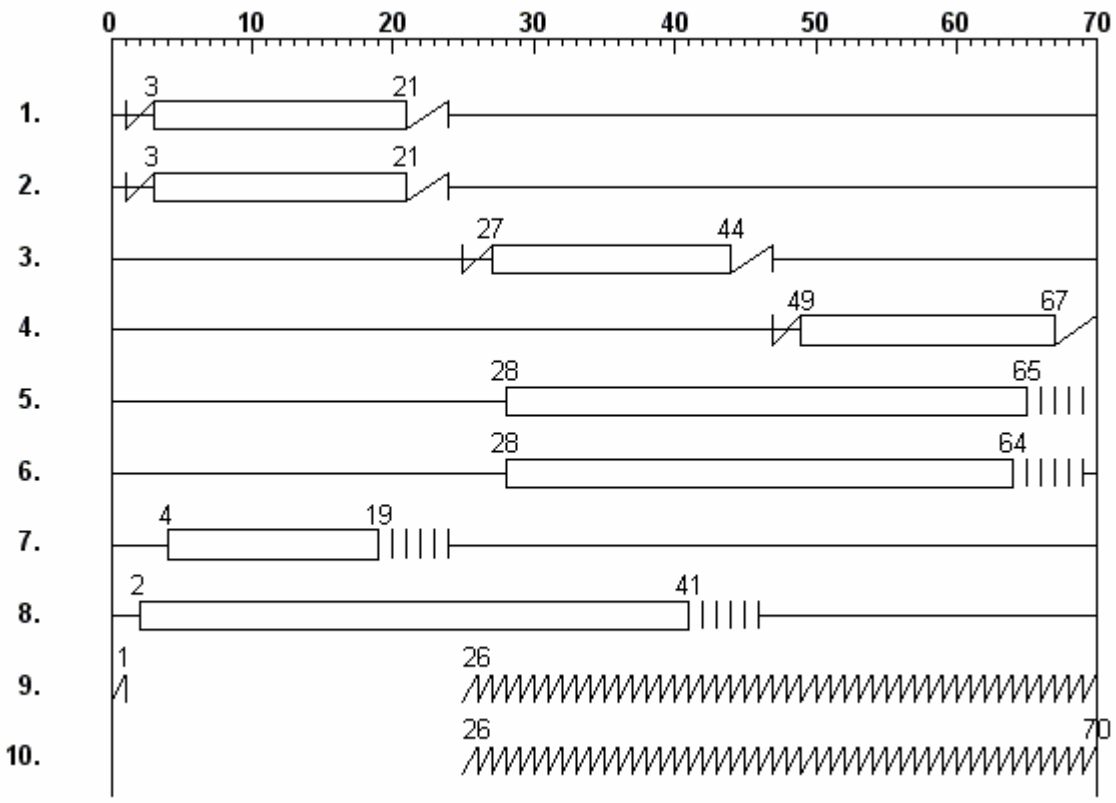
megjelenni. A kapott pontok, regressziós függvény (spline) segítségével lesznek összekötve.

5.2.2.3. Ftrajz.m

Az elkészített rendszer egyik sajátossága, hogy a kiszámított fázistervet képes grafikusan is ábrázolni. Ehhez azonban szükség van bizonyos adatokra:

- A kiszámított, zöldidő kezdetét és végét tartalmazó mátrixra
- Az egyes jelzőcsoportok típusát tartalmazó mátrixra
- Valamint az alkalmazott periódusidőre

Az ábra elkészítésekor egyszerű grafikai elemek, alakzatok (vonalak és téglalapok) kerültek felhasználásra. Az előállított vizuális tervre példa, a 14. ábrán látható.



14. ábra: Az aktuális adatok alapján létrehozott forgalomtervi ábra

5.2.3. A központi szoftver működésének összefoglalása

A szerver, funkcióit tekintve, egy forgalomszabályozási és egy felhasználói interfészre osztható. A forgalomszabályozási rész elsődleges eljárása a server, mely a hálózati adatkapcsolatért, az adott csomópontok forgalmának szabályozásáért, valamint a forgalomtechnikai adatok archiválásáért felel. Egy terepi berendezés jelentkezésekor, először azonosítja azt, majd betölti az adott készülékhez tartozó csomóponti adatokat a további szerver műveletek elvégzéséhez. Ezt követően a beolvasott kliens üzenetet átadja az AdatX függvénynek, mely a továbbiakban szükséges adatformátumra hozza azt. A berendezés által mért járműszámok alapján, három számítási eljárás, függvény is lefut:

- Periódusidő kiszámítása (peridoszam.m)
- Fázisok zöldidőinek kiszámítása (teloszt.m)
- Fázisterv megállapítása (ptervszam.m)

Ezt az adatok archiválása követi. Lementésre kerülnek a detektoradatok és a kiszámított fázisterv is. Erre a feladatra a mentes elnevezésű függvény hivatott.

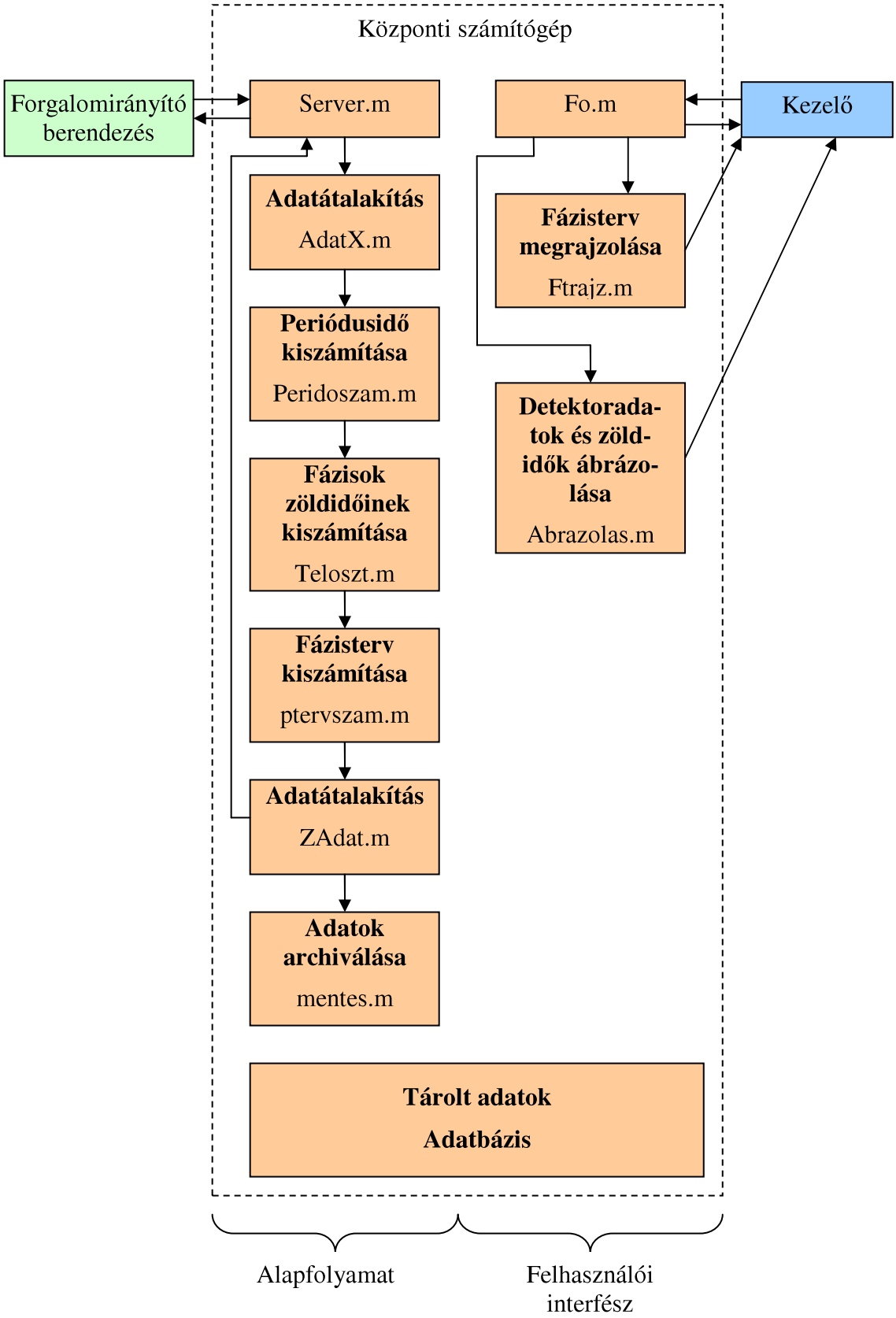
A műveletet zárásaként a megállapított fázisterv továbbításra kerül, a terepi berendezés felé, amit itt is adatkonvertálás (ZAdat.m) előz meg. Ezután kilépési feltétel vizsgálat következik, ha nincs probléma, akkor az egész folyamat kezdődik újra elölről.

A központi szoftver másik részét adó, felhasználói interfész szolgáltatásaihoz a főablakon (fo.m, fo.fig) keresztül lehet hozzáférni. Egy legördülő listából lehet választani a központi gép által kezelhető berendezések közül. A kiválasztást követően betöltődnek és különböző formákban (helyszínrajz, táblázatok, feliratok) megjelenítődnek az adott csomópontra vonatkozó adatok, valamint a fázisokra vonatkozó korlátozó adatok (minimum zöld, maximum zöld) is, amiket akár módosítani is lehet.

További adatlekérdezési funkciók is vannak, melyeket a főablakból lehet elérni:

- A fázisterv grafikus megjelenítését végző alprogram (ftrajz.m)
- Az archivált forgalomtechnikai adatok ábrázolása (abrazolas.m): Bizonyos paraméterek (dátum, detektor vagy járműcsoport, óra, adattípus) megadását követően négy fajta diagramot kaphatunk:
    - Forgalomnagyság napi alakulása (pontok, regressziós görbével)
    - Járműforgalom alakulása adott órában (oszlop diagram)
    - Átlagos zöldidők alakulása teljes napra (pontok, regressziós görbével)
    - Kiosztott zöldidők adott órára vonatkozóan (oszlopdiagram)

A szerver alkalmazás részei és az azok közötti adatforgalom a 15. ábrán is végig követhető.



15. ábra: A Központi szoftver felépítése és működése

6. Tesztelés

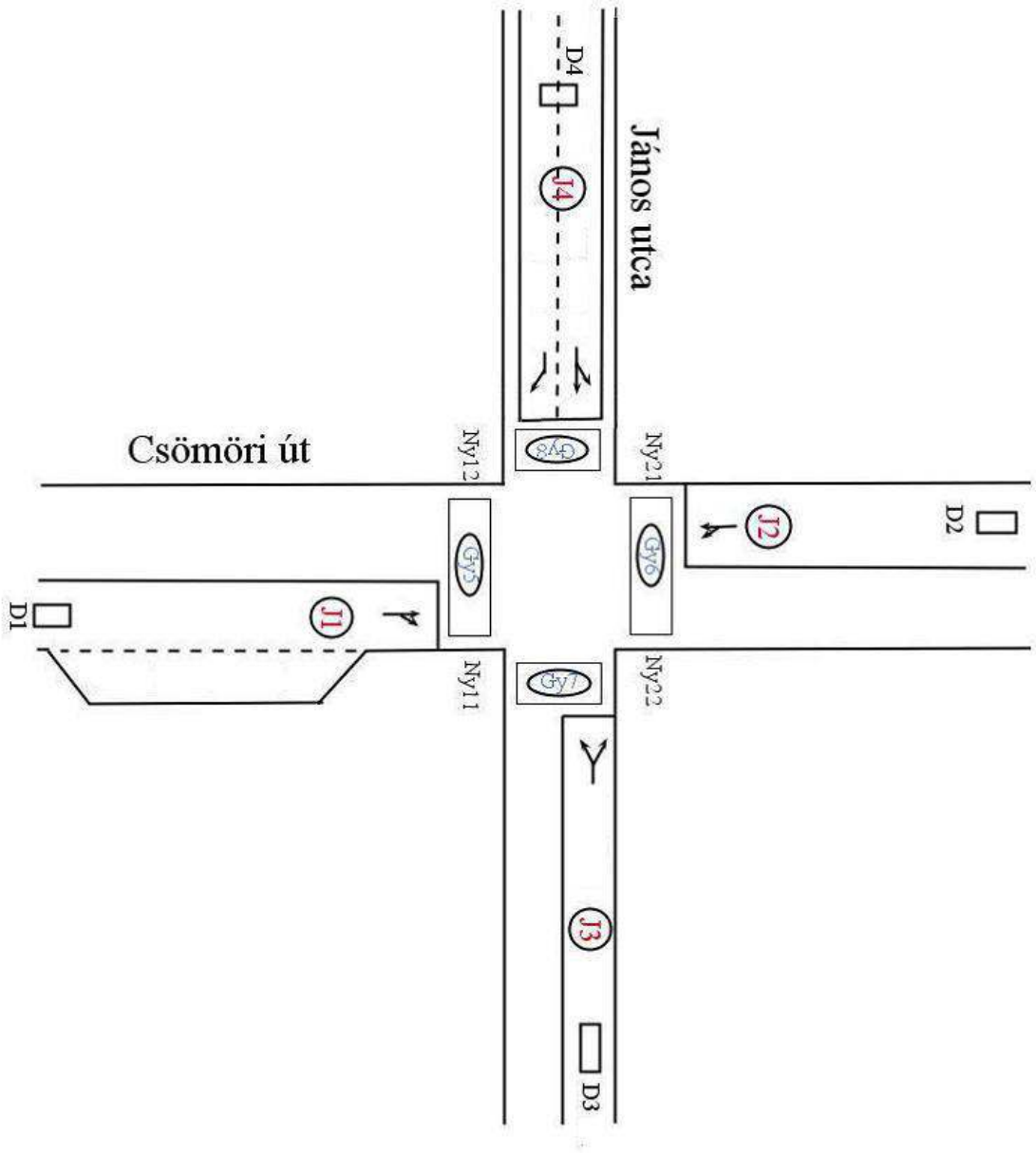
Az elkészült programok tesztelése a Budapesti Műszaki és Gazdaságtudományi Egyetem ZA07 laborjában került sor, ahol a Közlekedésautomatikai Tanszék által, e célból biztosított ACTROS VTC-3000-es készüléke is található, amely a jelen szakdolgozat egyik legfontosabb eleme, mivel az elkészített alkalmazások egy része erre a berendezésre készült.

16. ábra: A szerver és a kliens tesztelés közben

A kliens gép üzembelépése előtt a szervernek már üzemképes állapotban kell lennie, ellenkező esetben az előbbi kapcsolati hiba miatt leállítja az új funkciók ellátásáért felelős szálat és egy alternatív forgalomtechnikai programra áll át. A központi forgalomirányító szoftver egy terepi berendezés kéréséig „várakozó módban” van.

A szerver alapvető elérhetősége, amit a kliens gépeknek ismerniük kell: IP cím: 10.10.50.39, port szám: 11111.

A jelenlegi rendszer tesztelésekor csak egy darab készülék állt rendelkezésre. A berendezés típusa ACTROS VTC 3000, és a hozzákészített program, alapból a Csömöri út és János utca kereszteződéshez kapcsolódó jármű és gyalogosforgalmak lebonyolításáért felel. A csomópont egyszerűsített képe a 16. ábrán látható.



17. ábra: A Csömöri út és a János utca egyszerűsített ábrája

A szerverben tárolt adott csomópontra vonatkozó adatok:

Jelzőcsoport sorszáma	Jelzőcsoport típusa	Fázis	Detektor sorszáma	Villogó – gyalogos jelzőcsoport
1	Jármű	1	1	
2	Jármű	1	2	
3	Jármű	2	3	
4	Jármű	3	4	
5	Gyalogos	2		
6	Gyalogos	2		

7	Gyalogos	1		
8	Gyalogos	1		
9	Villogó	2		5
10	Villogó	2		6

3. Táblázat: Az adott csomópontra vonatkozó adatok

Ezen túlmenően, a számítások elvégzéséhez elengedhetetlen az adott csomópontra vonatkozó, közbensőidő mátrix is, mely a következő ábrán (17. ábra) is látható:

Behaladó

Kihaladó	1	2	3	4	5	6	7	8	9	10
1			6	6	6	7				
2			6	6	7	5				
3	6	5		5			5			
4	6	6	6				7	5		
5	8	6								
6	8	9								
7			8	6						
8				8						
9										
10										

18. ábra: A Csömöri út és a János utca csomópont közbensőidő mátrixa

A szabályozáshoz szükséges, adatok a 3. Táblázatban láthatók. Az itt leírt értékek 60 másodperces ciklusidőre vonatkoznak. (Ciklusidő változásnál a minimum értékek nem, de a maximum értékek arányosan változnak).

Fázis	Minimum	Maximum
1	10	20
2	5	15
3	5	15

4. Táblázat: A korlátozó adatok

Mivel a jelen berendezés csak oktatási célokat szolgál, tehát nem végez tényleges forgalomirányítási feladatokat, a detektoradatokat „mesterségesen” kell előállítani. Az adatok előállítása meghatározott feltételek mellett kell végbemenni, úgy, hogy azok valós állapot látszatát keltsék. Jelen tesztelésnél teljes napi (4 óra és 23 óra között) szimuláció lett végrehajtva, természetesen nem valósidőben. Ahhoz, hogy a szimuláció, minél inkább valósághű legyen, ezért a detektoradatok generálását a terepi berendezésnek kell elvégezni, ehhez azonban módosítani kell annak programját.

A változtatások:

Var.java

Ahhoz, hogy ne valós időben történjen a teljes napi vizsgálat, ezért az időpontok megfelelő kezeléséért egy naptár objektumok kell felvenni. Kezdő értéknek 2010.12.06 3 óra 59 perc 0 másodperc lett adva. Megjegyezendő, hogy itt a dátum teljesen irreleváns, csak az óra, perc és másodperc a fontos;

```
c=Calendar.getInstance();
c.set(2010, 12, 06, 03, 59, 00);
```

Atalakit.java

A módosítások nagy része ebben az osztályban történt. Először is a detektoradatok mérésért felelős addVerkehrsStaerke és getVerkehrsStaerke metódusok eltávolításra kerültek, helyükre a következőkben leírt kódok kerültek.

Első körben a *c* változóban tárolt időpont meg lett növelve a periódusidővel.

```
client.c.add(client.c.SECOND,+client.p);
```

Ezután következik, az adott órához tartozó járműszámok generálásához szükséges adatok megadása:

- Alap: alap járműérték, egyfajta minimum
- Plusz: az alap járműértékhez hozzáadandó érték
- Szorzo: Korábbról származó mérési adatok alapján, az egyes irányok járműforgalmának arányai a legnagyobb forgalmúéhoz képest.

```
for(i=0;i<client.detszam;++i){
  if (client.c.getTime().getHours()==4){alap=1;plusz=2;}
  if (client.c.getTime().getHours()==5){alap=2;plusz=2;}
  if (client.c.getTime().getHours()==6){alap=5;plusz=3;}
  if (client.c.getTime().getHours()==7){alap=9;plusz=3;}
  if (client.c.getTime().getHours()==8){alap=7;plusz=3;}
  if (client.c.getTime().getHours()==9){alap=6;plusz=3;}
  if (client.c.getTime().getHours()==10){alap=6;plusz=3;}
  if (client.c.getTime().getHours()==11){alap=5;plusz=3;}
  if (client.c.getTime().getHours()==12){alap=6;plusz=3;}
  if (client.c.getTime().getHours()==13){alap=6;plusz=3;}
  if (client.c.getTime().getHours()==14){alap=6;plusz=3;}
  if (client.c.getTime().getHours()==15){alap=7;plusz=3;}
  if (client.c.getTime().getHours()==16){alap=7;plusz=3;}
  if (client.c.getTime().getHours()==17){alap=7;plusz=3;}
  if (client.c.getTime().getHours()==18){alap=6;plusz=3;}
  if (client.c.getTime().getHours()==19){alap=4;plusz=3;}
  if (client.c.getTime().getHours()==20){alap=3;plusz=2;}
  if (client.c.getTime().getHours()==21){alap=2;plusz=2;}
  if (client.c.getTime().getHours()==22){alap=1;plusz=2;}
```

Végül az aktuális detektoradat kiszámlálása következik:

```
  client.x[i]=   (int)   Math.round((double)(alap*szorzo+Math.random()
*plusz)*((double)client.p/60));
```

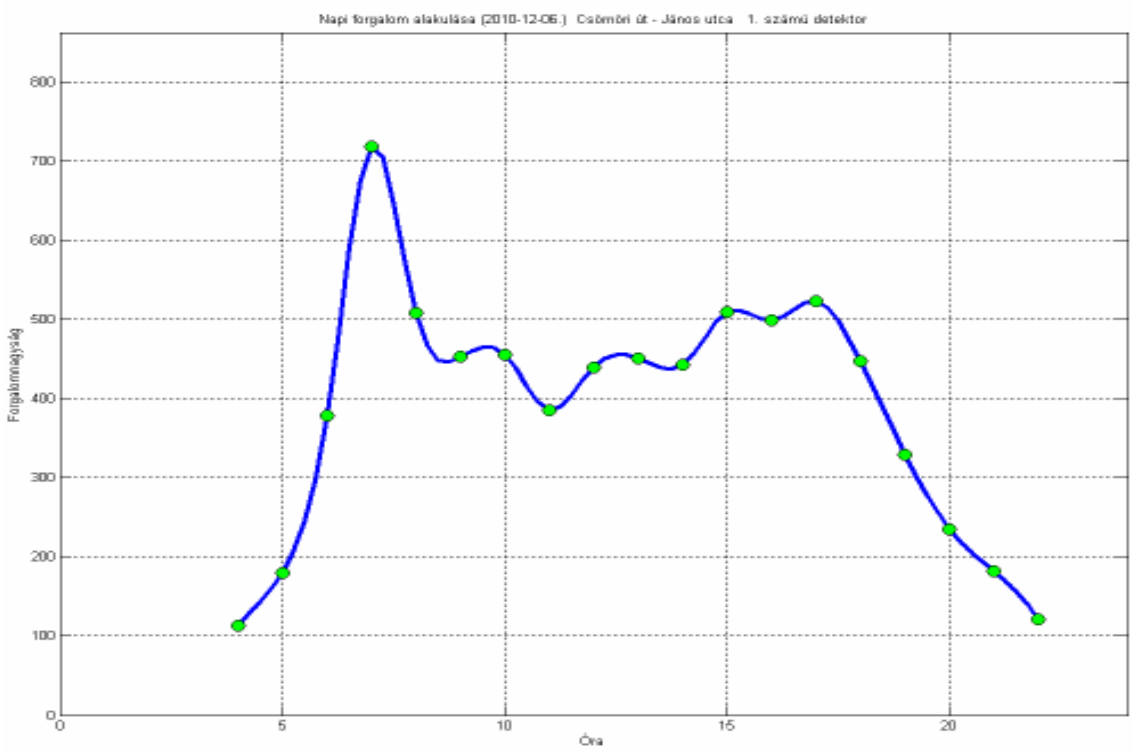
Ismetelt.java

Itt csak egy egyszerű adatmódosítás történt, a kliens programrész lefutásának ciklusidejét szabályzó rész átírásával:

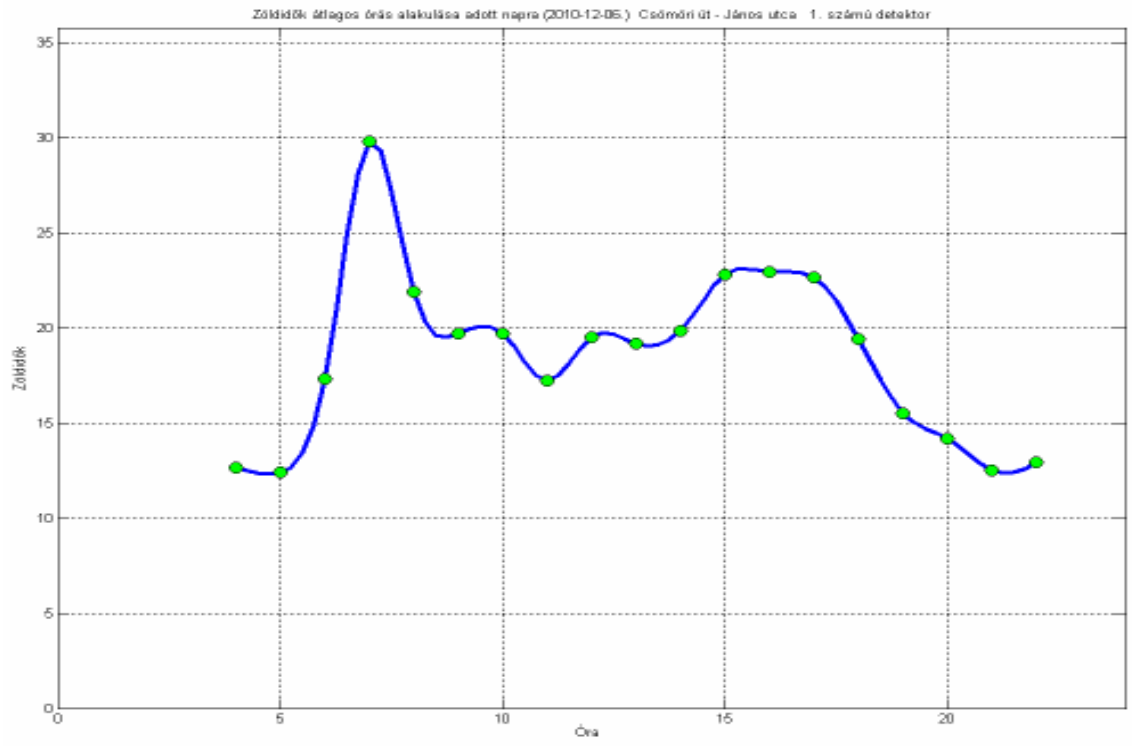
```
Thread.sleep(500);
```

Így a teszt adatok küldése fél másodpercenként történt, ezáltal a teljes szimuláció néhány perc alatt zajlott le.

A tesztelés során kapott eredményeket a következő ábrákon lehet meg tekinteni, ahol a 19. a 20. és a 21. ábra, az 1. számú jelzőcsoporthoz tartozó járműves irányra vonatkoznak.

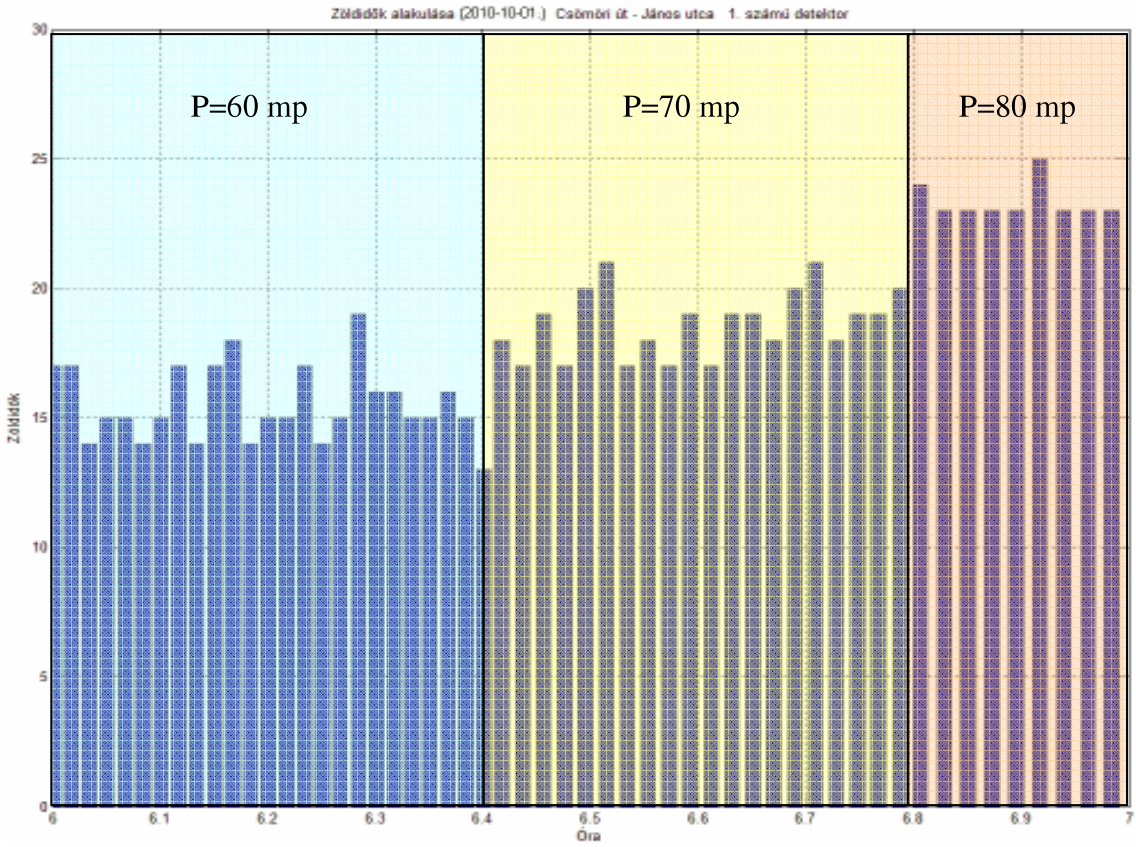


19. ábra: 1. számú jelzőcsoporthoz tartozó járműves irány napi forgalmának alakulása



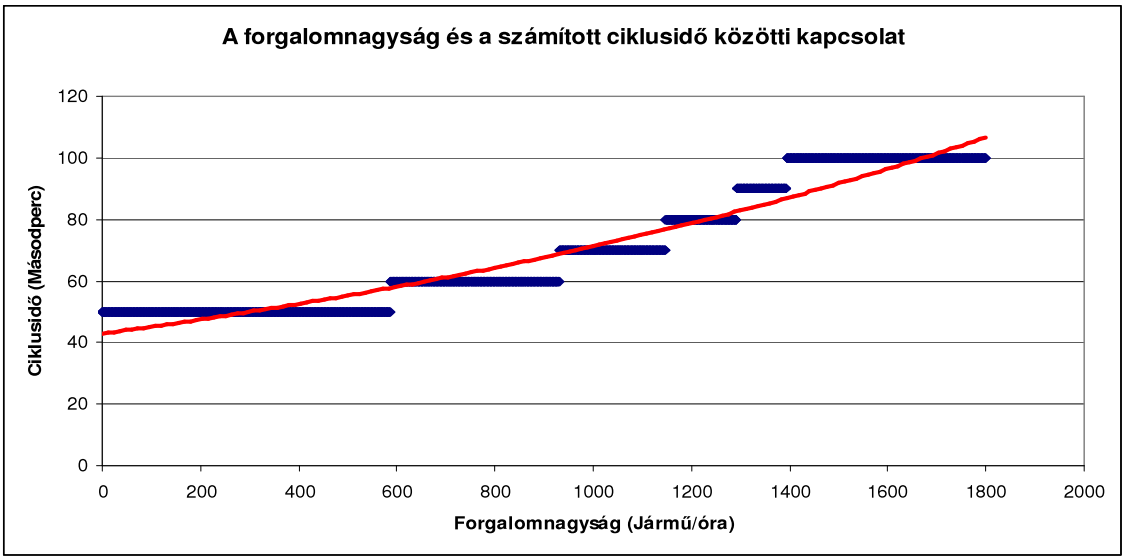
20. ábra: az 1. számú jelzőcsoporthoz tartozó járműves iránynak kiosztott átlagos zöld-időinek alakulása teljes napra vonatkozóan

A következő ábrán a zöldidők alakulását láthatjuk adott időpontban (6 és 7óra kötött), jelölve a ciklus idők változását is



21. ábra: az 1. számú jelzőcsoporthoz tartozó járműves iránynak, 6 és 7 óra között kiosztott zöldidőinek alakulása, az alkalmazott ciklusidők jelölésével

A ciklusidő megállapítására használt képlet alapján, a ciklusidő és a forgalomnagyság közötti kapcsolat a következő képen alakul:



22. ábra: A forgalomnagyság és a ciklusidő közötti kapcsolat

7. Továbbfejlesztési lehetőségek

Az elkészített rendszer képes ellátni a feladatát, tehát az új központ, a rácsatlakoztatott készülékek által küldött adatok alapján, képes új forgalmi tervet készíteni az éppen aktuális közlekedési állapot, valamint a korlátozó feltételek figyelembe vételével. A szakdolgozat keretében kialakított központi szoftver és az ahhoz módosított ACTROS forgalomirányító berendezést vezérlő program, természetesen nem vetekedhet a manapság használatban lévő alkalmazásokkal, megvalósításokkal, viszont kiindulási alapnak megfelelő.

A továbbfejlesztési kérdést szoftveres, illetve hardveres, illetve szerver és kliens oldalról is meg lehet közelíteni.

Mivel, a jelen rendszernek egyik legfontosabb jellemzője az internet alapú kommunikáció, ezért nagy hangsúlyt kapnak az adatbiztonsági kérdések. Gyors és hatékony adatkódolásra van szükség, a külső adatmanipulációk elkerülése végett. A biztonság fokozása érdekében tanácsos elkülönülni a nyilvános rendszertől, amit virtuális magán hálózatok kialakításával vagy akár különálló, internet és telefonszolgáltatóktól bérelt hálózatokkal oldható meg.

Vezeték nélküli internet alkalmazásával komoly kábelezési költségek takaríthatók meg viszont ez hardveres és szoftveres beavatkozásokat igényel a terepi eszköz és a központ esetében is, valamint újabb biztonsági kérdések merülnek fel.

A létrehozott rendszer bár forgalomfüggő, de jelen esetben, bizonyos eseményekre (pl.: gyalogosjelzők aktiválása, buszbejelentkezés) érzéketlen. A hatékonyság növelése érdekében, érdemes a rendszer ilyen irányú továbbfejlesztése is, az alkalmazott algoritmusok bővítésével.

A központi számítógép teljesítményétől függően, a rajta futtatott szoftver jelen helyzetben egy kliens kérést a másodperc törtrésze (a teszt alapján egy középkategóriás laptopnak ez 0,2 másodperc) alatt tudja kezelni. Ez elég gyors, akár több tíz berendezés gond nélküli kiszolgálásához (egy komolyabb tudású számítógép esetében). A probléma több kezelendő terepi készülék esetén van, mert a jelen megvalósításban, a központ egy időben csak egy kéréssel foglalkozik, így a kérések ütközhetnek. Erre megoldás lehet, hogy az egyes kérések külön program-szálon kerüljenek kidolgozásra, így egy időben több kérés-kezelés is történhet.

További fejlesztések ajánlottak a szerver-oldali kezelő alkalmazásoknál is. Növelni kell a szabályozható jellemzők számát. Ezen kívül, új beavatkozási lehetőségek is kerülhetnek a rendszerbe (pl.: különleges eseményekkor egyes irányok hosszabb idejű tiltása, vagy engedélyezése, programok kivezérlése, stb.). A felhasználó felé nyújtott információk mennyiségét is lehetne növelni:

- A rendszerelemek diagnosztikai információval
- Valósidejű adat és jelzéskép lekérdezés
- A meglévő adatlekérdező és ábrázoló funkciók bővítése

A rendszer önállóságát is lehet lehetne fokozni, új funkciók beépítésével mint például:

- irányok dinamikus fázisba sorolása
- fázissorrend változtatása
- ellenőrző, beavatkozó és felülbíráló funkciók bekerülése
- stb.

Tanácsos a meglévő adatbázis kivitelezést és kezelést korszerűbb megvalósításra cserélni, a biztonságosabb adatkezelést és rugalmasabb adatlekérdezést megcélozva. Az egyik leghatékonyabb megoldás: SQL adatbázis-kezelés

Kényelmesebb és rugalmas hozzáférést és távfelügyeleti munkát, webes hozzáférési felületek kialakításával lehetne megoldani. Ezáltal az adatlekérdezések és beavatkozások már nem lennének helyhez kötve. Ilyen esetekben foglakozni kell a felhasználóazonosítással és a jogosultságok kezelésével is.

8. Összefoglalás

Ezen szakdolgozat egy irányítási rendszer, egy lehetséges megvalósításának megtervezéséről és kivitelezésével foglalkozott.

A dolgozat első részében a forgalomirányító rendszerekről és azok elemeiről (központ, terepi berendezés) volt szó. A rendszerek felépítése lehet központos, elosztott, illetve vegyes struktúrájú. Ez alapján a rendszer elemeinek is más és más a szerepe, ami elsősorban a döntésmeghozatalának helyében nyilvánul meg, ugyanis, a jelzésképek megállapítása történhet egy központban, vagy helyileg, egy intelligens forgalomirányító berendezésben, mint például a jelen dolgozatban kiemelten foglakozott, ACTROS VTC-3000-ben.

Az esetek legnagyobb részében a központ és a csomóponti berendezések között kapcsolat van, a közöttük lévő adatcsere megvalósításáért különböző kommunikációs protokollok felelnek. Ilyen például a Magyarországon még most is elterjedt BEFA, vagy az OCIT, mely a TCP/IP alapú protokollon alapul. A jelen fejlesztései és elgondolásai, egy olyan jövőképet vizionálnak, melyben a központ elegendő információval és erőforrással van ellátva, ahhoz, hogy az irányítási feladat nagy részét elvégezze, úgy, hogy a terepi berendezések, szinte teljesen elhagyhatóak. Az egyes eszközök vezérlése, a velük történő adatcsere általános célú megoldásokon (ethernet) alapul.

Ettől a ponttól, a dolgozat az új rendszerrel foglalkozik. Először leírásra kerültek a kidolgozandó rendszer elemei és tulajdonságai, amik nagyvonalakban a következők: A központi számítógép IP alapú kapcsolaton keresztül megkapja a terepi berendezés által, a detektorok felől összegyűjtött járműszám-adatokat, majd ez és a felhasználó által megadott feltételek alapján elkészíti az adott csomópont fázistervét. A kiszámított terv ezután visszakerül a csomóponti készülékhez, ami néhány pillanaton belül már alkalmazza is azt. Ezek alapján a rendszert értelmezhetjük is úgy, mint szerver-kliens kapcsolatot is, így a leírás két nagyobb egységre bontható, egy kliens és egy szerver folyamatokat leíróra.

A kliens, vagyis közúti forgalomirányító berendezés ACTROS VTC 3000 készülék, melynek a már meglévő programja került módosítása, illetve kiegészítésre. A berendezést JAVA nyelven lehet programozni, ami előnyt jelent más forgalomirányító készülékkel szemben, melyeknél speciális programnyelv van használatban. A meglévő részek új funkciókkal lettek kiegészítve, melyek biztosítják a szerverrel történő adatkapcsolatot és az ehhez szükséges adatkonverziókat, valamint a visszaérkezett fázisterv alkalmazását.

A szerver, mely egy általános számítógép is lehet, speciális, az irányítási feladatok ellátását szolgáló szoftvert futtat. A MATLAB fejlesztői környezetben készített központi alkalmazás is két nagyobb egységre bontható: egy a szerver feladatokat ellátó részre, mely az adatforgalomért, a számításokért és az adatok archiválásáért felel, valamint egy grafikus felhasználói interfészre, ami felhasználóbarát módon biztosítja korlátozó feltételek megadását, módosítását, valamint a mentett forgalomtechnikai adatok grafikon formában történő megjelenítését.

Az alkalmazások részletes leírását követően, a rendszer működőképességét bizonyító tesztelési esetleírás következett, ahol egy nem-valósidejű teljes, napos szimuláció eredményeit lehetett grafikonok formájában végigkövetni és a működésre vonatkozó követ-

keztetéseket levonni. Zárásképp, a rendszer esetleges tovább fejlesztésének lehetséges módszereiről volt pár szó.

Felhasznált irodalom

Elektronikus dokumentumok

[1] Tettamanti Tamás, Luspay Tamás, Dr. Varga István: Közúti közlekedési automatika. Budapest, http://www.kka.bme.hu, 2008. 9. oldal. 2.1 A forgalomirányító rendszer általános felépítése.

[2] Tettamanti Tamás, Luspay Tamás, Dr. Varga István: Közúti közlekedési automatika. Budapest, http://www.kka.bme.hu, 2008. 125. oldal. 6.1.3 Irányítási stratégiák

[3] Markos Papageorgiou: Review of Road Traffic Control Strategies

[4] Tettamanti Tamás, Luspay Tamás, Dr. Varga István: Közúti közlekedési automatika. Budapest, http://www.kka.bme.hu, 2008. 57. oldal. 4.2 A Nikola Tesla forgalomirányító berendezés

[5] Tettamanti Tamás, Luspay Tamás, Dr. Varga István: Közúti közlekedési automatika. Budapest, http://www.kka.bme.hu, 2008. 59. oldal. 4.3 FB016 forgalomirányító berendezés

[6] Siemens 800 http://www.mobility.siemens.com/shared/data/pdf/www/infrastructure_logistics/plus_technology_for_sitraffic_c800_en.pdf

[7] Tettamanti Tamás, Luspay Tamás, Dr. Varga István: Közúti közlekedési automatika. Budapest, http://www.kka.bme.hu, 2008. 87. oldal. 4.6 Irányítási stratégiák

[8] Tettamanti Tamás, Luspay Tamás, Dr. Varga István: Közúti közlekedési automatika. Budapest, http://www.kka.bme.hu, 2008. 100. oldal. 4.9 Irányítási stratégiák

[9] Vilati-Signalbau Huber: ACTROS VTC 3000 teljes dokumentáció (német), 2009.

[10] Tettamanti Tamás: ACTROS VTC 3000 Segédlet a Közúti irányító és kommunikációs rendszerek II. c. tárgyhoz. Budapest, http://www.kka.bme.hu ,2010.

[11] Ethernet dokumentáció: http://www.cisco.com/en/US/docs/internetworking/technology/handbook/Ethernet.html

[12] Budapesti Műszaki és Gazdaságtudományi Egyetem, Irányítástechnika és Informatika Tanszék, Magyarországi Terepbusz központ: http://www.fsz.bme.hu/traficc/profibus/profibus_index.html

[13] Profibus hivatalos honlap: http://www.profibus.com/

[14] Terepbusz: http://www.fsz.bme.hu/traficc/can/can.html

[15] CAN hivatalos honlap: http://www.can-cia.org/

[16] Terepbusz: http://www.fsz.bme.hu/traficc/asibus/as_index.html

[17] AS-Interface hivatalos honlap:http://www.as-interface.co.uk/

[18] Tettamanti Tamás, Luspay Tamás, Dr. Varga István: Közúti közlekedési automatika. Budapest, http://www.kka.bme.hu, 2008. 125. oldal. 6.1.3 Irányítási stratégiák

[19] OCIT hivatalos honlap: http://www.ocit.org/

[20] Mathworks hivatalos honlap http://www.mathworks.com/products/matlab/

[21] Java hivatalos honlap: http://www.oracle.com/technetwork/java/index.html

[22] Nagy Gusztáv: JAVA programozás. Kecskemét 2007. http://nagygusztav.hu

[23] Eclipse hivatalos honlap: http://www.ecpise.org

[24] Polgár János, Tettamanti Tamás: Forgalomtechnikai kód az ACTROS VTC 3000 forgalomirányító berendezésben. Budapest XVI. Csömöri út-János utca csomópont jelzésterveinek működése alapján. Budapest, http://www.kka.bme.hu, 2010.

Folyóirat

[25] Dr. Varga István, Közúti jelzőlámpák vezérlése hálózaton keresztül.

Felhasznált alkalmazások

Programozás

- Eclipse: http://www.ecpise.org
- Matlab: http://www.mathworks.com/products/matlab/

Egyéb, segédalkalmazások

- Putty. Nyílt forráskódú, telnet SSH kliens. Ingyenesen letölthető: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
- Filezilla. Fájlok, FTP kapcsolaton keresztüli fel és letöltését szolgáló program. Ingyenesen letölthető: http://filezilla-project.org

Ábrajegyzék

Melléklet - Programkódok

Kliens programkódok

Var.java

```
package csojan;
…
public class Var
{

…
      public static ForgfProgX prog5;
      public static ForgfProgX prog6;
      public static ForgfProgX prog7;
      public static ForgfProgX prog8;
      public static ForgfProgX prog9;
      public static ForgfProgX prog10;

…
      public static     int          detszam     =4;
      public static     int          jcsszam     =10;
      public static     int[]        x           =new int[detszam];
      public static     int[][]      zkv         =new int[jcsszam][2];
      public static     boolean      allj        =false;
      public static     Detektor[]   d           =new Detektor[detszam];
      public static     int          p;

…
}
```

Init.java

```
package csojan;
…
public class Init
    implements Initialisierung
{
…
    protected void initialisiereDet()
    {
        for(int i=0;i<Var.detszam;i++){
            Var.d[i]= new Detektor(Var.tk1,"Dt"+i,0,0,0,i+12);
        }
…
}

    protected void initialisiereProgs()
    {
…
       Var.prog5=new ForgfProgX(Var.tk1, "P5", 5, 50, 3, 3, 0, 0);
       Var.prog6=new ForgfProgX(Var.tk1, "P6", 6, 60, 3, 3, 0, 0);
       Var.prog7=new ForgfProgX(Var.tk1, "P7", 7, 70, 3, 3, 0, 0);
       Var.prog8=new ForgfProgX(Var.tk1, "P8", 8, 80, 3, 3, 0, 0);
       Var.prog9=new ForgfProgX(Var.tk1, "P9", 9, 90, 3, 3, 0, 0);
       Var.prog10=new ForgfProgX(Var.tk1, "P10", 10, 100, 3, 3, 0, 0);
```

```
…
}
    protected void initialisiereHW()
    {
…
        for(int i=0;i<Var.detszam;i++){
            new IoKanal(Var.d[i],io1,1+i);}
…
}
…
    public static void main(String args[])
    {
…
      Ismetelt ism=new Ismetelt();
      ism.start();
…
}
```

Ismetelt.java

```
package csojan;
import java.io.IOException;

public class Ismetelt extends Thread
{
      public void run(){
            while (Var.allj!=true){
                  try {
                        Thread.sleep(30000);
                  } catch (InterruptedException e1) {
                        e1.printStackTrace();
                  }
                  try {
                        AdatBeKi.ABK();
                        try {
                              Thread.sleep(Var.p*1000);
                        } catch (InterruptedException e) {
                              e.printStackTrace();
                        }
                  }catch (IOException e) {
                        e.printStackTrace();
                   }
            }
      }
}
```

AdatBeKi.java

```
package csojan;
import java.io.*;
import java.net.*;

public class AdatBeKi {
      public static     String      servercim   ="10.10.50.37";
      public static     int         serverport  =11113;
      public static     Socket      csomag      =null;
      public static     BufferedReader    be    =null;
      public static     PrintWriter       ki    =null;
```

```
public static void ABK()

throws IOException{

            try{
                  csomag=new Socket(servercim,serverport);
                  be = new BufferedReader(new
InputStreamReader(csomag.getInputStream()));
                  ki=new PrintWriter(csomag.getOutputStream());

                  ki.println(Atalakit.DetAdat());
                  ki.flush();

                  Atalakit.AdatZold(be.readLine());

                  csomag.close();
                  be.close();
                  ki.close();

            }catch(UnknownHostException e){}

             catch(IOException e){  Var.allj=true;}


}
}
```

Atalakit.java

```
package csojan;
import vt.StgEbene;

public class Atalakit {
      public static String DetAdat(){
            int i;
            String ertek      =     "";
            String adat       =     "";

            for(i=0;i<Var.detszam;++i){
                  Var.x[i]=Var.d[i].getVerkehrsStaerke();
                  ertek=Integer.toString(Var.x[i]);
                  if (ertek.length()==1) {ertek="00"+ertek;}
                  else if (ertek.length()==2) {ertek="0"+ertek;}
                  adat+=ertek;
      }
      return adat;
}

      public static void AdatZold(String sz){
            for(int i=0;i<Var.jcsszam;++i){
                  for(int j=0;j<2;++j){

      Var.zkv[i][j]=Integer.parseInt(sz.substring(i*6+j*3,i*6+j*3+3));
}
            }
            Var.p=Integer.parseInt(sz.substring(sz.length()-
3,sz.length()));

            for(int i=0;i<Var.jcsszam;++i){
```

```
                    if (Var.zkv[i][0]-2<0)
                          Var.zkv[i][0]+=Var.p-2;
                    else Var.zkv[i][0]-=2;

              for(int i=0;i<Var.detszam;i++){
                      Var.d[i].addVerkehrsStaerke(2*Var.p);}

       if (Var.p==50)
                    Var.tk1.setProgWunsch(Var.prog5,
StgEbene.STG_VT_ANWENDER);
            else if (Var.p==60)
                    Var.tk1.setProgWunsch(Var.prog6,
StgEbene.STG_VT_ANWENDER);
      else if (Var.p==70)
             Var.tk1.setProgWunsch(Var.prog7, StgEbene.STG_VT_ANWENDER);
      else if (Var.p==80)
             Var.tk1.setProgWunsch(Var.prog8, StgEbene.STG_VT_ANWENDER);
      else if (Var.p==90)
             Var.tk1.setProgWunsch(Var.prog9, StgEbene.STG_VT_ANWENDER);
      else if (Var.p==100)
             Var.tk1.setProgWunsch(Var.prog10,StgEbene.STG_VT_ANWENDER);
       }

}
```

ForgfProgX

```
package csojan;

import sbh.vtbib.vtc3000.K;
import sg.Sg;
import vt.*;


public class ForgfProgX extends LogikProg
{
    public ForgfProgX(TeilKnoten arg0, String arg1, int arg2, int
arg3, int arg4, int arg5, int arg6,
            int arg7)
    {
        super(arg0, arg1, arg2, arg3, arg4, arg5, arg6, arg7);
    }

    public ForgfProgX(TeilKnoten arg0, String arg1, int arg2, Sg
arg3[], int arg4, int arg5, int arg6,
            int arg7, int arg8)
    {
        super(arg0, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8);
    }

    public void programmFunktion()
    {
            K.BEKI(Var.j11, Var.zkv[0][0], Var.zkv[0][1]);
            K.BEKI(Var.j21, Var.zkv[1][0], Var.zkv[1][1]);
            K.BEKI(Var.j31, Var.zkv[2][0], Var.zkv[2][1]);
            K.BEKI(Var.j41, Var.zkv[3][0], Var.zkv[3][1]);
            K.BEKI(Var.gy51, Var.zkv[4][0], Var.zkv[4][1]);
            K.BEKI(Var.gy61, Var.zkv[5][0], Var.zkv[5][1]);
            K.BEKI(Var.gy71, Var.zkv[6][0], Var.zkv[6][1]);
```

```
            K.BEKI(Var.gy81, Var.zkv[7][0], Var.zkv[7][1]);
            K.BEKI(Var.sv91, Var.zkv[8][0], Var.zkv[8][1]);
            K.BEKI(Var.sv101, Var.zkv[9][0], Var.zkv[9][1]);
        }
}
```

Szerver programkódok

Server.m

```
function server
    import java.net.*
    import java.io.*
    import java.lang.*

    kilep=false;
    szamlalo=0;

    global foirany;
    global szunet;

    zkv=[];
    P=50;
    szunet=[];

    global csp;
    global id;
    global aktualis;

    load('csp.mat','csp');

    disp('------------------------------------------------------------');
    disp('Szerver program elindult. Inicializálás...');

    while(kilep==false)

      disp('Várakozás egy kliens gép kérésére...')

      scsomag=ServerSocket(11111);
      csomag=scsomag.accept;

      tic

      be=BufferedReader(InputStreamReader(csomag.getInputStream));
      ki=PrintWriter(csomag.getOutputStream,true);

      cim=char(csomag.getInetAddress());
      cim=cim(2:length(cim));
      aktualis=0;
      for i=1:length(csp)
          if csp(i).ipcim==cim;
              aktualis=i;
              break;
          end;
      end;
```

```
        disp(datestr(now,'yyyy.mm.dd.  HH:MM:SS'));
        disp(['Beérkező kérés a(z) ' char(cim) ' felől.']);
        disp(['Csomópont: ' csp(aktualis).nev]);

        keres=readLine(be);
        detektor_adatok=AdatX(keres);
        disp(['Kérés: ' char(keres)]);

        P=peridoszam(P,detektor_adatok);
        disp(['Periódusidő megállapítása. P= ' num2str(P) ' mp.']);

        load ('adatok.mat','adatok');
        zik=adatok(aktualis).zik;
        zik(:,2)=round(zik(:,2)*P/60);

        tz=P-sum(szunet);

        disp('Számítás...');
        fazis_z=teloszt(tz,zik,detektor_adatok);
        zkv=ptervszam(fazis_z,P);

        adatok(aktualis).zkv=zkv;
        adatok(aktualis).p=P;
        save('adatok.mat','adatok');

        valasz=ZAdat(zkv,P);
        ki.println(valasz);
        disp(['Válasz: ',valasz]);

        disp('Adatok mentése. Bejegyzés a naplóba.');
        mappa1=['Archivum\' csp(aktualis).nev '\DetAdat\'];
        mappa2=['Archivum\',csp(aktualis).nev,'\Zold\'];
        mappa3='Naplo\';
        file=[datestr(now,'yyyy-mm-dd'),'.txt'];
        mentes(mappa1,file,keres);
        mentes(mappa3,file,cim);
        mentes(mappa2,file,valasz);

        disp('Kapcsolat bontása.');
        csomag.close;
        scsomag.close;
        be.close;
        ki.close;

        t=toc;

        disp(['Művelet elvégésének ideje: ' num2str(t) ' mp']);
        disp('------------------------------------------------------------
----------------');

    end

    disp('Szerver program leállítása.');

end
```

AdatX.m

```
function x = AdatX(sz)  %a paraméterként kapott string (=járműszamok)
tartalmát egy új mátrixba helyezi;

    for i=0:(length(sz)/3-1)
        x(i+1)=str2num(substring(sz, i*3,i*3+3));
    end
end
```

ZAdat.m

```
function adat = ZAdat(zkv,P)

    function uj=kiegeszit(regi)
        switch (length(regi))
            case (1)
                regi=['00' regi];
            case (2)
                regi=['0' regi];
        end
        uj=regi;
    end


  disp(zkv);
  adat='';
  for i=1:length(zkv(:,1))
      for j=1:2
        ertek=num2str(zkv(i,j));
        ertek=kiegeszit(ertek);
        adat=[adat ertek];
      end
  end

  adat=[adat kiegeszit(num2str(P))];


end
```

Peridoszam.m

```
function P=peridoszam(p_akt,detektor_adatok)

    global csp;
    global aktualis;

    global foirany;
    global szunet;

    fazisszam=max(csp(aktualis).fazis);

    for i=1:fazisszam
```

```
        maxertek(i)=-1;
        foirany(i)=0;
        szunet(i)=0;
    end

    for i=1:length(csp(aktualis).det)
        if ( detektor_adatok(i)>
maxertek(csp(aktualis).fazis(csp(aktualis).det(i))) )
            maxertek(csp(aktualis).fazis(csp(aktualis).det(i)))=
            detektor_adatok(i);
            foirany(csp(aktualis).fazis(csp(aktualis).det(i)))=
            csp(aktualis).det(i);
        end
    end

    for f=1:fazisszam
        kovf=f+1;
        if (kovf>fazisszam)
              kovf=1;
        end
        szunet(f)=csp(aktualis).kozb(foirany(f),foirany(kovf));
    end

    P=((120*sum(szunet))/(1-((sum(maxertek)*3600/p_akt)/1800)))^(1/2);
    P=round(P/10)*10;

    if (P>100)
       P=100;
    end;

    if (P<50)
        P=50;
    end;

end
```

Teloszt.m

```
function z = teloszt (tz,zik,x)

    tzk=tz;
    tzi=tz;
    n=length(zik(:,1));

    benne=false;

    for i=1:n
        megvan(i)=0;
    end

    ell=sum(x);
    if (ell==0)
        ell=1;
    end
    osszk=ell;
    osszi=osszk;
```

```
while(benne==false)
    benne=true;
    for i=1:n
        if (~(megvan(i)==1))
           z(i)=round(tzk/osszk*x(i));
        end
    end

    for i=1:n
         if (z(i)<zik(i,1))
            z(i)=zik(i,1);
            tzi=tzi-z(i);
            megvan(i)=1;
            benne=false;
            osszi=osszi-x(i);
        end

        if (z(i)>zik(i,2))
            z(i)=zik(i,2);
            tzi=tzi-z(i);
            megvan(i)=1;
            benne=false;
            osszi=osszi-x(i);
       end
    end

    tzk=tzi;
    osszk=osszi;

end


kul=tz-sum(z);

    if (kul==1)
        nagy=0;
        xn=0;
        for i=1:n
            if (x(i)>xn && z(i)+kul<=zik(i,2))
                nagy=i;
                xn=x(nagy);
            end
        end
        z(nagy)=z(nagy)+kul;
    end

    if (kul==-1)
        kicsi=0;
        xk=inf;
        for i=1:n

            if (x(i)<xk && z(i)+kul>=zik(i,1))
                kicsi=i;
                xk=x(kicsi);
            end
        end
        z(kicsi)=z(kicsi)+kul;
    end
```

```
        if (kul>1)
            while (kul~=0)
              for i=1:n
                  if (z(i)<zik(i,2))
                      z(i)=z(i)+1;
                      kul=kul-1;
                      if (kul==0)
                          break;
                      end
                  end
              end
            end
        end

end
```

Ptervszam.m

```
function zkv=ptervszam(z,p)

global csp;
global aktualis;

global foirany;
global szunet;

kezdo=3;

fazisszam=max(csp(aktualis).fazis);
iranyszam=length(csp(aktualis).kozb(:,1));

for s=1:iranyszam
    for o=1:2
        zkv(s,o)=-1;
    end
end


for i=1:fazisszam
    if (i==1)
        zkv(foirany(i),1)=kezdo;
    else
        zkv(foirany(i),1)=zkv(foirany(i-1),2)+szunet(i-1);
    end
    zkv(foirany(i),2)=zkv(foirany(i),1)+z(i);

        if (zkv(foirany(i),1)>p)
            zkv(foirany(i),1)=zkv(foirany(i),1)-p;
        end
        if (zkv(foirany(i),2)>p)
            zkv(foirany(i),2)=zkv(foirany(i),2)-p;
        end

        if (zkv(foirany(i),1)<0)
            zkv(foirany(i),1)=zkv(foirany(i),1)+p;
        end
        if (zkv(foirany(i),2)<0)
```

```
            zkv(foirany(i),2)=zkv(foirany(i),2)+p;
        end
end


for i=1:iranyszam
    if ( (csp(aktualis).tipus(i)<5) && (zkv(i,1)==-1) )
        fazis_a=csp(aktualis).fazis(i);

        fazis_e=fazis_a-1;
        if (fazis_e==0)
            fazis_e=fazisszam;
        end


          fazis_k=fazis_a+1;
          if (fazis_k>fazisszam)
              fazis_k=1;
          end


          rendben=false;

          for k=1:iranyszam
              if ( (csp(aktualis).kozb(i,k)~=0) &&
(csp(aktualis).fazis(k)==fazis_k) && (csp(aktualis).tipus(k)<5) )
                  rendben=true;
                  break;
              end
          end

          if (rendben==false)
              zkv(i,2)=zkv(foirany(fazis_a),2);
          else


        % zöld vég
            for j=1:iranyszam
                    if ( (zkv(j,1)~=-1) &&
(csp(aktualis).fazis(j)==fazis_k) )
                      if (zkv(i,2)==-1)
                          zkv(i,2)=zkv(j,1)-csp(aktualis).kozb(i,j);
                      else
                        zkv(i,2)=max(zkv(i,2),zkv(j,1)-
csp(aktualis).kozb(i,j));
                      end
                    end
            end

          end


          rendben=false;

          for k=1:iranyszam
              if ( (csp(aktualis).kozb(k,i)~=0) &&
(csp(aktualis).fazis(k)==fazis_e) && (csp(aktualis).tipus(k)<5) )
                  rendben=true;
```

```
                    break;
                end
            end

            if (rendben==false)
                zkv(i,1)=zkv(foirany(fazis_a),1);
            else
            %zöld kezdet


              for j=1:iranyszam
                if ( (zkv(j,2)~=-1) && (csp(aktualis).fazis(j)==fazis_e)
)
                    if (zkv(i,1)==-1)
                        zkv(i,1)=zkv(j,2)+csp(aktualis).kozb(j,i);
                    else

zkv(i,1)=max(zkv(i,1),zkv(j,2)+csp(aktualis).kozb(j,i));
                    end
                end
              end
            end

          %korrigálás
          if (zkv(i,1)>p)
              zkv(i,1)=zkv(i,1)-p;
          end
          if (zkv(i,2)>p)
              zkv(i,2)=zkv(i,2)-p;
          end

          if (zkv(i,1)<0)
              zkv(i,1)=zkv(i,1)+p;
          end
          if (zkv(i,2)<0)
              zkv(i,2)=zkv(i,2)+p;
          end

      end

  end


  for i=1:iranyszam
      if ( (csp(aktualis).tipus(i)==5) && (zkv(i,1)==-1) )
          fazis_a=csp(aktualis).fazis(i);

          fazis_e=fazis_a-1;
          if (fazis_e==0)
              fazis_e=fazisszam;
          end

          rendben=false;

          while (rendben==false)
            fazis_k=fazis_a+1;
            if (fazis_k>fazisszam)
                fazis_k=1;
            end
```

```
            if(fazis_a==fazis_k)
                break;
            end

            for k=1:iranyszam
                if ( (csp(aktualis).kozb(i,k)~=0) &&
(csp(aktualis).fazis(k)==fazis_k) )
                    rendben=true;
                    break;
                end
            end

             if (rendben==false)
                fazis_a=fazis_k;
            end

        end

        %zöld kezdet
        for j=1:iranyszam
            if ( (zkv(j,2)~=-1) && (csp(aktualis).fazis(j)==fazis_e) )
                if (zkv(i,1)==-1)
                    zkv(i,1)=zkv(j,2)+csp(aktualis).kozb(j,i);
                else

zkv(i,1)=max(zkv(i,1),zkv(j,2)+csp(aktualis).kozb(j,i));
                end
            end
        end

        % zöld vég
        for j=1:iranyszam
            if ( (zkv(j,1)~=-1) && (csp(aktualis).fazis(j)==fazis_k) )
                if (zkv(i,2)==-1)
                    zkv(i,2)=zkv(j,1)-csp(aktualis).kozb(i,j);
                else
                    zkv(i,2)=min(zkv(i,2),zkv(j,1)-
csp(aktualis).kozb(i,j));
                end
            end
        end

        %korrigálás
        if (zkv(i,1)>p)
            zkv(i,1)=zkv(i,1)-p;
        end
        if (zkv(i,2)>p)
            zkv(i,2)=zkv(i,2)-p;
        end

        if (zkv(i,1)<0)
            zkv(i,1)=zkv(i,1)+p;
        end
        if (zkv(i,2)<0)
            zkv(i,2)=zkv(i,2)+p;
        end
    end

end
```

```
 if (size(csp(aktualis).villogo)~=[0 0])
    for i=1:length(csp(aktualis).villogo(:,1))

zkv(csp(aktualis).villogo(i,2),1)=zkv(csp(aktualis).villogo(i,1),1)-2;

zkv(csp(aktualis).villogo(i,2),2)=zkv(csp(aktualis).villogo(i,1),2)+6;
    end
 end

zkv=zkv+csp(aktualis).offset;
for i=1:iranyszam
        if (zkv(i,1)>p)
            zkv(i,1)=zkv(i,1)-p;
        end
        if (zkv(i,2)>p)
            zkv(i,2)=zkv(i,2)-p;
        end

        if (zkv(i,1)<0)
            zkv(i,1)=zkv(i,1)+p;
        end
        if (zkv(i,2)<0)
            zkv(i,2)=zkv(i,2)+p;
        end
end

end
```