Compact dynamisch busstation

A.S. Klusener, S.F.M. van Vlijmen, A. Schrijver

Compact dynamic

Computer Science/Department of Software Technology

Note CS-N9601 May 1996

CS

CWI is the National Research Institute for Mathematics and Computer Science. CWI is part of the Stichting Mathematisch Centrum (SMC), the Dutch foundation for promotion of mathematics and computer science and their applications.
SMC is sponsored by the Netherlands Organization for Scientific Research (NWO). CWI is a member of ERCIM, the European Research Consortium for Informatics and Mathematics.

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 0169-118X

Compact Dynamisch Busstation

A.S. Klusener (CWI)
S.F.M. van Vlijmen (UU)
A. Schrijver (CWI)
Universiteit Utrecht, Centrum voor Wiskunde en Informatica

Abstract

In opdracht van Nederland Haarlem hebben de Universiteit Utrecht (UU) en het Centrum voor Wiskunde en Informatica (CWI) een studie gemaakt naar het ontwerp van een Compact Dynamisch Busstation (CDB).

Het betreft een regelsysteem voor het dynamisch toewijzen van bussen aan perrons, waarbij constant ingespeeld wordt op afwijkingen van de dienstregeling.

Deze studie heeft geleid tot een model van een Compact Dynamisch Busstation; zodra er vertragingen optreden waardoor reserveringen van bussen met elkaar in conflict raken, wordt er met behulp van lineaire programmeringstechnieken een nieuwe optimale toewijzing opgesteld.

Ook is er een zgn. prototype ontwikkeld, een software omgeving waarin dit model is uitgewerkt, zodat men zich ook een beeld kan vormen van het daadwerkelijk functioneren.

AMS Subject Classification (1991): 68Q45, 68Q60, 68Q65, 68Q90.
CR Subject Classification (1991): D.2.1, D.2.2, D.2.6, F.2, F.4.
Keywords & Phrases: Compact Dynamisch Busstation, lineaire programmeringstechnieken, ToolBus.

Inleiding

Wat is een Compact Dynamisch Busstation?

Een conventioneel busstation kent voor elke buslijn een vast perron. Als verschillende lijnen op hetzelfde perron geplaatst zijn, dan moet er een marge bestaan tussen hun onderlinge passeertijden om conflicten te vermijden in het geval een van de beide bussen te laat aankomt. Dit leidt tot relatief grote busstations, die op elk moment gedeeltelijk leeg zijn.

Een *compact dynamisch busstation* is een meer efficiënt busstation (vandaar *compact*), waarbij het toewijzen van een bus aan een perron *dynamisch* gebeurt.

In onze opzet wordt er van alle ritten die binnen een bepaalde looptijd vallen, bijv. een dag, een reservering van een perron bijgehouden. Op het moment dat een rit vertraagd blijkt te zijn, wordt zijn reservering verlengd. Zo'n verlenging kan conflicten tussen verschillende gereserveerde ritten veroorzaken, waarna het noodzakelijk is om bepaalde ritten aan een ander perron toe te wijzen. Deze nieuwe toewijzing moet zo optimaal mogelijk gekozen worden, waarvoor de besliskundige techniek *lineair programmeren* wordt gebruikt.

Volgens dit mechanisme is er voor elke binnenkomende bus een perron gereserveerd, waar de bus op kan plaatsnemen.

Context van het project

Het project betreft het ontwerp van een Compact Dynamisch Busstation, uitgevoerd door de vakgroep Toegepaste Logica van de Universiteit Utrecht (UU) en de afdelingen Besliskunde en Programmatuur van het Centrum voor Wiskunde en Informatica (CWI), voor Nederland Haarlem (NH), nationaal marktleider op het gebied van de verkeersregelsystemen.

De UU en CWI streven naar het toepassen van hun onderzoeksresultaten op het gebied van (formele) software-ontwerp en -analyse en van besliskunde in industriële toepassingen.

NH heeft UU/CWI bij hun CDB-project betrokken om na te gaan of met geavanceerde technieken een innovatief product als het CDB gerealiseerd kan worden.

Aanvankelijk lagen de problemen vooral op het gebied van het modelleren en de besliskunde. Later, toen de nadruk meer op de ontwikkeling van het prototype kwam te liggen, betrof het vooral een test-case m.b.t. de Toolbus ([BK95]).

Met enige regelmaat is er contact geweest tussen UU/CWI en NH, in het bijzonder met Jan Kroone, Hans van Ruyven en Henk Spronk. Ook is Hans Goossens van het onderzoeks- en adviesbureau D&P, waarmee NH samenwerkt op het gebied van het CDB, een keer bij een dergelijke bijeenkomst geweest. Deze bijeenkomsten zijn voor de auteurs van groot belang geweest om een helder beeld van het probleem te verkrijgen.

De auteurs willen Prof. Paul Klint bedanken voor zijn hulp m.b.t. de Toolbus, en Adri Steenbeek voor enkele aanwijzingen aangaande Cplex.

Contents

1 Een kort overzicht van de inhoud van het stuk

In dit stuk worden alle benodigde begrippen ingevoerd, evenals hun onderlinge samenhang. Omdat geabstraheerd is van een aantal aspecten die bij een concreet station een rol kan spelen, spreken we van een *model*.

1.1 Het kern–model

Eerst wordt in Sectie 2 een zgn. *kern–model* ingevoerd, dat een aantal basisbegrippen bevat die hoogstwaarschijnlijk een rol spelen bij elk mogelijk concreet compact dynamisch busstation. Deze begrippen worden hieronder kort aangehaald, de nummers (1.1, 1.2, e.d.) geven de secties aan waarin ze verder worden uiteengezet.

- 2.1. Het *station*, bestaande uit *perrons*, *buffers*, *ingangen* en *uitgangen*.
- 2.2. Intuïtief kent een busstation lijnen die met enige regelmaat een bepaalde dienst rijden. Bijv. een lijn 4, die van 7.00 tot 19.00 om tien over heel en tien over half vertrekt.

  Elke keer dat een bus een dienst van een lijn rijdt, noemen we een *rit*. We gaan er in dit document van uit dat in een zgn. *perrontoewijzing*, of kortweg *toewijzing*, per rit de volgende informatie wordt bijgehouden:

  - Een uniek *ritnummer*,
  - het *toegewezen perron*,
  - het *voorkeurs-perron*,
  - het *begintijdstip* van de reservering,
  - de *lengte* van de reservering,
  - de *halteertijd* (kleiner of gelijk aan de lengte van de reservering),
  - de *status* van de reservering,
  - het *lijnnummer*,
  - de *route* en
  - de *dienstwagen*,

  Gedurende de loop van de tijd worden alleen het *toegewezen perron*, de *status* en de *lengte* van de reservering aangepast, de andere gegevens blijven ongewijzigd. Het *toegewezen perron* is initieël gelijk aan het *voorkeurs-perron*. In het vervolg zullen beide noties uitgebreid aan de orde komen.

  De gegevens in een perrontoewijzing kunnen afdwingen dat de ene rit moet plaatsnemen achter een andere, en eventueel na verloop van tijd moet oprijden naar een volgende plek op hetzelfde perron. De posities van de ritten op de perrons noemen we *haltes*. In sommige situaties kan het van pas komen om deze halte-informatie expliciet te hebben. Daarom wordt in Sectie 2.3 de notie van een perrontoewijzing uitgebreid tot die van een *haltetoewijzing*, waarbij voor elke rit ook wordt opgegeven hoelang de rit op de verschillende posities moet staan.

- 2.4 Een toewijzing wordt *consistent* genoemd als er geen reserveringen met elkaar in conflict zijn, anders dan spreken we van een *inconsistente* toewijzing. Zo is een toewijzing inconsistent als er teveel ritten tegelijkertijd, in de zin dat de totale lengte van de dienstwagens de lengte van het perron overschrijdt, aan hetzelfde perron worden toegewezen.

- 2.5 Als een toewijzing inconsistent is, dan moet er een nieuwe toewijzing opgesteld worden, die zo *optimaal* mogelijk is. Hiertoe wordt een zgn. *kostenfunctie* opgesteld die aan elke mogelijke toewijzing kosten toekent. Een toewijzing is dan optimaal als de berekende kosten minimaal zijn.

  In de huidige opzet is een toewijzing optimaal als de ritten zo dicht mogelijk bij hun zgn. referentie-perron geplaatst worden (zie volgend punt).

- 2.6 De *status* speelt een belangrijke rol in de wijze waarop de rit een rol speelt in de kostenfunctie; de status bepaalt of het *referentie*-perron het *voorkeurs-perron* is, dan wel het *toegewezen perron*. Uit de status valt verder af te lezen of een rit al of niet spoedig zal aankomen, zoja dan wordt het (laatst) toegewezen perron als referentie-perron genomen (omdat het dan immers onwenselijk om de rit nog te verplaatsen), zonee, dan wordt het voorkeurs-perron genomen (omdat de rit, als het enigszins kan, in de buurt van zijn voorkeurs-perron geplaatst dient te worden).

  Ook bepaalt de status hoe zwaar de afstand tot het *referentie*-perron in de kostenfunctie meetelt, op deze wijze kan worden afgedwongen dat bij voorkeur ritten worden verplaatst die voorlopig nog niet binnenkomen.

- 2.7 *Detecties* worden verricht d.m.v. zgn. detectielussen die in het wegdek zijn aangebracht. We gaan er gemakshalve van uit dat als een dienstwagen gedetecteerd wordt, dat dan ook bekend is welke dienstwagen dat is en welke rit hij rijdt.

  Vlak voor een rit bij het station aankomt zal hij eerst *gedetecteerd* worden op een zgn. toevoerweg, hiertoe zijn toevoerwegen uitgerust met detectoren, op deze wijze wordt een rit aan het systeem *vooraangekondigd*.

  Van elke aankomende rit is bekend langs welke toevoerweg hij binnenkomt, en hoe laat hij dus naar verwachting vooraangekondigd zou moeten worden. Als op een bepaald moment een rit nog niet vooraangekondigd is terwijl dat volgens de planning wel had moeten gebeuren, dan kan worden geconcludeerd dat de rit vertraagd is, waarna de reservering van deze rit wordt verlengd.

  Bussen worden ook gedetecteerd bij het aankomen op, en het verlaten van, het station.

  In de status van een rit wordt bijgehouden waar de rit gedetecteerd is, of dat de rit vertraagd is.

- 2.8 Als een bus moet oprijden naar een perron, of naar een buffer, of moet vertrekken van een perron, dan wordt dit d.m.v. een *instructie* (op een scherm) aangegeven.

  Bij het geven van deze instructies is het gegarandeerd dat ritten in de goede volgorde op een perron geplaatst worden, en dat een bus pas oprijdt wanneer er voldoende ruimte op het betreffende perron beschikbaar is.

- 2.9 De *looptijd* is de termijn, bijv. een dag, waarbinnen de ritten worden bijgehouden in de toewijzing. Ook wordt er een (tijds)-*horizon* bijgehouden, alle ritten die binnen het huidige tijdstip en deze horizon vallen krijgen een hogere prioriteit dan de "latere" ritten.

1.2 Uitbreidingen op het kern–model

Vervolgens worden in Sectie 3 een aantal uitbreidingen op dit kern–model geïntroduceerd, zodat het verder ingevuld kan worden tot een meer specifiek model van een compact dynamisch busstation. Voorbeelden van dergelijke uitbreidingen zijn:

- 3.1 Uitsluiten van bepaalde toewijzingen, omdat bijv. bepaalde ritten niet op bepaalde perrons mogen of kunnen halteren.
- 3.2 Het *reserveperron*.
- 3.3 *Correlatie* tussen verschillende ritten bij het opstellen van een volgende, optimale, toewijzing.
- 3.4 Indraai– en uitdraai–ruimte, om aan te geven dat een rit voor het indraaien of uitdraaien op resp. van een bepaald perron een zekere vrije ruimte vereist op een ander perron.
- 3.5 Het is principe ook mogelijk dat een bepaalde strikte eis genuanceerder gesteld wordt als 'de eis moet zo min mogelijk overschreden worden'.

1.3 Het prototype

Nadat het kern–model en enkele mogelijke uitbreidingen zijn ingevoerd, wordt in Sectie 4 een prototype behandeld, dat enigszins is gebaseerd op de gegevens van het station Apeldoorn [DOWA94]. Dit prototype betreft een uitwerking van het kern–model en de uitbreidingen uit de Secties 3.1 en 3.2. Het bestaat uit de volgende componenten:

- Een *Initialisator*, die één voor één de overige componenten opdraagt zich te initialiseren.
- Een *Controller*, die de wijzigingen n.a.v. de laatste detecties doorvoert (vertragingen, veranderingen van status etc.), en vervolgens de benodigde instructies geeft.
- Een *Consistent-maker*, die nagaat of een toewijzing wel consistent is, en zo niet bepaalde ritten zo optimaal mogelijk op een ander perron plaatst.

  In Sectie 5 zal een klein voorbeeld gegeven worden van het opstellen van de benodigde lineaire ongelijkheden.
- Een *Grafische-Interface*, die de toewijzing van bijv. het komende uur grafisch afbeeld. Ook wordt per perron de informatie gegeven over de ritten die eraan zijn toegewezen.

  Deze interface wordt na elke wijziging opnieuw ingevuld.
- Een *Instructor*, die de instructies afbeeldt.
- Een *Klok*, die met een bepaalde frequentie, bijv. om de 15 sec. een puls geeft.

Deze componenten zijn geïntegreerd m.b.v. de Toolbus, zie Sectie 4.8.

2 Het kern–model

2.1 De indeling van het station

In dit stuk bestaat een compact dynamisch busstation uit, zie ook Figuur 2.1:

- één of meer *perrons*, een perron is een locatie waar één of meer bussen op kunnen halteren om passagiers in en/of uit te laten stappen. Er wordt vanuit gegaan dat bussen elkaar niet kunnen inhalen op een perron.

  De volgorde van de bussen op een perron wordt bijgehouden d.m.v. *haltes*, dus halte 1, halte 2 etc.. Merk op dat als de bussen een verschillende lengte hebben, dat perrons dan geen vast aantal haltes hebben.

- nul of meer buffers, een buffer is een locatie waar een bus gestald kan worden, er kunnen geen passagiers in- of uitstappen op een buffer.

- één of meer ingangen, een ingang heeft een detector, een ingangsdetector, waarmee kan worden vastgesteld welke bus het station oprijdt.

- één of meer uitgangen, een uitgang heeft een detector, een uitgangsdetector, waarmee kan worden vastgesteld welke bus het station verlaat.

- één of meer toevoerwegen die elk een detector hebben, een toevoerdetector. De toevoerdetector dient om een nauwkeurige schatting te kunnen maken van de aankomsttijd van een bus bij een ingang.

- een bord (voor bijv. passagiers) waarop de meest recente informatie staat afgebeeld.

- een bord (voor bijv. chauffeurs) waarop instructies staan afgebeeld, die betrekking hebben op het plaatsnemen in een buffer, het oprijden naar en het verlaten van een perron.

In Sectie 3 wordt een aantal uitbreidingen behandeld, zoals bijv. een reserveperron.

2.2 De perrontoewijzing

Een perrontoewijzing bevat voor elke rit de volgende informatie:

- Een uniek *ritnummer*.

- Het *toegewezen perron*.

  Dit is het perron dat voor de rit gereserveerd is. Het kan afwijken van het voorkeursperron als deze voorkeurs-reservering in conflict is met reserveringen van andere ritten, en de rit daarom niet op zijn voorkeursperron geplaatst kan worden.

- Het *voorkeursperron*.

  Dit is het perron waar de rit in principe op geplaatst zal worden; mocht dat niet mogelijk zijn, dan zal de rit zo dicht mogelijk bij dit perron geplaatst worden.

- Het *begintijdstip* van de reservering.

- De *lengte* van de reservering.

  Deze wordt elke keer verlengd als de rit vertraagd blijkt te zijn.

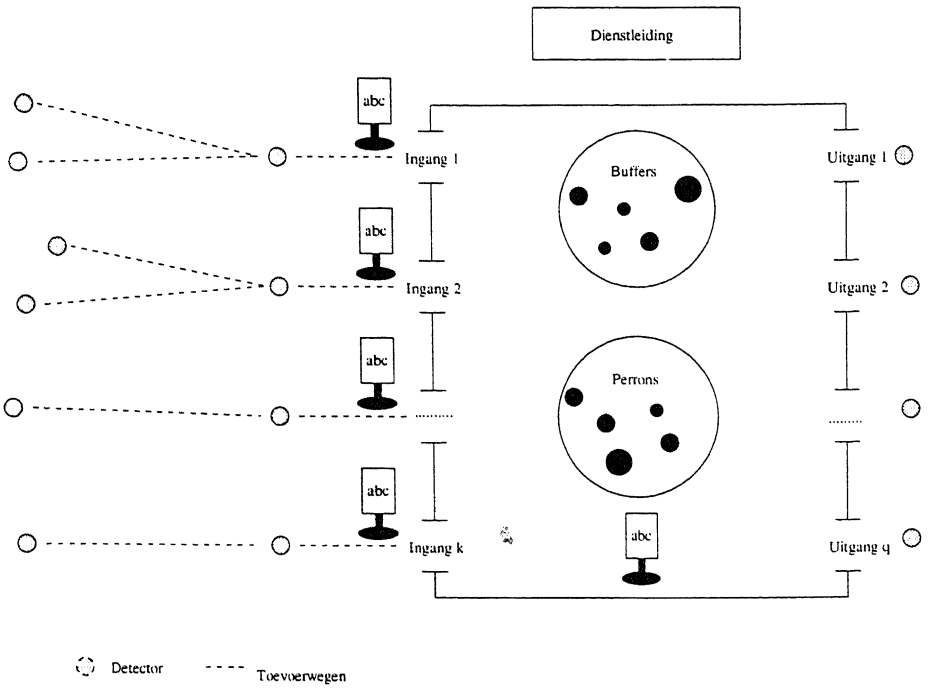

Figure 1: Een schematische voorstelling van een CDB.

- De *halteertijd* (kleiner of gelijk aan de lengte van de reservering).
- De *status* van de reservering.

  Op zich zijn de waarden die deze status kan aannemen geen vast onderdeel van het model. Om een indruk te geven van de mogelijkheden van deze notie, volgt hieronder een aantal waarden en hun intuïtie, zoals die ook in het prototype zullen worden gehanteerd.

  - Over het algemeen is de status initieel *gereserveerd1* (de dienstregeling kan echter in principe ook een status als *urgent* of *VIP* aan een rit toekennen), zodra het begintijdstip van zo'n rit minder dan, bijv., 15 minuten verwijderd is, dan wordt de status "verhoogd" tot *gereserveerd2*, zie 2.9. Dit uit zich o.m. in een kleurverschil bij het afdrukken van de toewijzing door het user-interface.
  - Zolang de rit vertraagd is, voor de vooraankondiging, dan is de status *vertraagd.*
  - Als de rit wordt gedetecteerd op toevoerweg T, dan wordt de status *vooraangekondigd-T.* Als de rit bij ingang I gedetecteerd wordt, dan wordt de status *gearriveerd-I.*
  - Als vervolgens de rit de instructie krijgt op om buffer b, dan wel op het aan hem toegewezen perron plaats te nemen, dan wordt de status *op-buffer-b* resp. *op-perron.*
  - Als de rit vetrekt van het station, via uitgang U, dan wordt de status *vertrokken-U.*
  - Andere denkbare waarden voor de status zijn *uitgevallen*, *urgent* etc. .

- Het *lijnnummer.*

Het lijnnummer wordt vooral gebruikt bij het afbeelden van informatie aan de reizigers, voor wie het ritnummer immers te specifiek is.

- De *route*. De route bepaalt langs welke toevoerweg de rit naar verwachting binnenkomt, wat van belang is bij het bepalen of de rit al of niet vertraagd is.
- De *dienstwagen*.

  De dienstwagen is van belang, omdat deze bepaalt hoeveel ruimte een rit in beslag neemt.

Merk op dat de dienstregeling die aan de reizigers aangeboden wordt een veel meer abstracte en compacte representatie zal hebben. Zo worden het ritnummer, de verwachte aankomsttijd, de route en de dienstwagen veelal niet in de reizigersinformatie opgenomen, en wordt slechts aangegeven dat een lijn elk half uur, om 10 over heel en 10 over half, vertrekt. In dit document zullen we niet verder op dit verschil ingaan.

2.3 De haltetoewijzing

In een perrontoewijzing wordt vastgelegd op welk perron een rit is toegewezen, maar het legt niet expliciet de volgorde vast van de bussen die tegelijkertijd op eenzelfde perron gereserveerd zijn. Echter, bij het afdrukken van de informatie aan de reizigers en bij het geven van oprijd-instructies aan bussen, is het ook van belang de volgorde van de bussen op de perrons te weten. Immers, de passagiers moeten weten of ze in de eerste, of de tweede bus moeten stappen, en een bus die later vertrekt moet ook later oprijden.

Voor het vaststellen van zo'n volgorde is het noodzakelijk om voor elke rit na te gaan welke andere ritten er tegelijkertijd ook aan het perron zijn toegewezen. Een *halte*toewijzing is een uitbreiding van een *perron*toewijzing, waarin ook de volgordes van de ritten wordt bijgehouden. Stel een perrontoewijzing bevat o.m. de volgende informatie (alle andere gegevens als voorkeursperron, status e.d. zijn weggelaten):

	toegewezen perron	begin reservering	lengte reservering
Rit1:	1	10.00	5
Rit2:	1	10.02	5
Rit3:	1	10.02	8

Dan kan deze perrontoewijzing uitgebreid worden tot een haltetoewijzing door per rit een lijst van paren toe te voegen. Elk paar bevat een zgn. *halte-nummer* en een tijdstip dat aangeeft tot wanneer de rit op de halte moet staan. Deze lijst noemen we ook het *verloop* van de rit.

Als we aannemen dat op perron 1 rond 10 uur geen andere ritten gereserveerd zijn, dan kunnen we bovenstaande perrontoewijzing alsvolgt uitbreiden tot een haltetoewijzing:

	toegewezen perron	begin reservering	lengte reservering	verloop
Rit1:	1	10.00	5	(1,10.05)
Rit2:	1	10.02	5	(2,10.05),(1,10.07)
Rit3:	1	10.02	8	(3,10.05),(2,10.07),(1,10.10)

Deze haltetoewijzing geeft aan dat *Rit1* direkt kan oprijden naar halte 1, en daar om 10.05 van vertrekt; *Rit2* staat tot 10.05 op halte 2, en schuift dan door naar halte 1; *Rit3* staat aanvankelijk tot 10.05 op de derde halte, schuift dan door naar halte 2, waarna het om 10.07 opschuift naar halte 1, en om 10.10 van deze halte vertrekt.

In een meer realistische uitwerking is het misschien wenselijk om het doorschuiven te beperken; immers wanneer de reizigers al aan het instappen zijn kan een bus niet zonder meer gaan rijden. Dan is het vereist dat een rit niet meer vlak voor zijn vertrek nog doorschuift naar de volgende halte. Dit aspect zal echter buiten beschouwing gelaten worden.

Merk op dat alleen een consistente perrontoewijzing gegarandeerd uitgebreid kan worden tot een haltetoewijzing; de volgende ritten zijn met elkaar in conflict (*Rit5* komt eerder aan, maar gaat later weg!) en kunnen daarom niet in een volgorde geplaatst worden:

	toegewezen perron	begin reservering	lengte reservering
Rit4:	2	10.02	7
Rit5:	2	10.00	10

2.4 Consistente toewijzingen

Een toewijzing is consistent als aan de volgende eisen is voldaan:

- op elk moment dient de totale lengte van de dienstwagens van alle ritten die op dat moment op eenzelfde perron gereserveerd zijn, de lengte van dat perron niet te overschrijden.
- Als een rit een reservering heeft van 10.00 tot 10.07, met een halteertijd van 5 minuten, dan is het interval van 10.00-10.02 het *aankomst-interval* en van 10.05-10.07 het *vertrek-interval*. Het aankomst-interval resp. het vertrekt-interval is de marge waarbinnen de rit aankomt resp. vertrekt.

  Er is een tweetal eisen dat betrekking heeft op de onderlinge aankomst- resp. vertrek-intervallen van een paar ritten:

  - hun aankomst en vertrek-intervallen mogen onderling niet overlappend zijn (anders is hun volgorde van binnenkomst niet op voorhand bekend),
  - als het aankomst-interval van de ene rit voor het aankomst-interval van de andere ligt, dan moeten hun vertrek-intervallen op dezelfde wijze geordend zijn.

    Ofwel, een rit die eerder aankomt, moet ook eerder vertrekken.

  Merk op dat hoe groter deze marges, d.w.z. het verschil tussen de lengte van de reservering en de halteertijd, hoe minder combinaties van ritten op hetzelfde perron zijn toegestaan.

Indien na een wijziging als gevolg van een vertraging een toewijzing niet meer consistent is, dan zal een aantal ritten op een ander perron geplaatst moeten worden, en wel zodanig dat de nieuwe toewijzing wél weer consistent is.

Stel dat perron 4 slechts één bus kan bevatten, terwijl *Rit1* en *Rit2* er tegelijkertijd op geplaatst zijn:

	toegewezen perron	voorkeursperron	begin reservering	lengte reservering
Rit1:	perron 4	perron 6	10.10	5
Rit2:	perron 4	perron 4	10.12	5

dan moet één van de beide ritten op een ander perron gereserveerd worden, bijv. *Rit1* van perron 4 naar perron 5:

	toegewezen perron	voorkeursperron	begin reservering	lengte reservering
Rit1:	perron **5**	perron 6	10.10	5
Rit2:	perron 4	perron 4	10.12	5

2.5 De kostenfunctie en optimalae toewijzingen

Op zich zijn er vele mogelijkheden om een toewijzing weer consistent te maken, het model kiest uit al deze mogelijkheden de *optimale* volgens een van te voren vastgestelde *kostenfunctie*. Een kostenfunctie kent aan elke toewijzing bepaalde kosten, ofwel strafpunten, toe; de toewijzing met de minste kosten/strafpunten is de meest optimale. Dit wordt bepaald door voor elke rit de afstand te bepalen tussen het perron waarop die toewijzing de rit reserveert, en het referentieperron van de rit.

Een mogelijkheid om de kosten van een toewijzing op te stellen, is om voor elke rit deze afstand te bepalen, en vervolgens al deze verkregen afstanden op te tellen. Voor *Rit1*, die perron 6 als referentieperron heeft, komen we dan tot de volgende reeks van strafpunten (stel er zijn 9 perrons):

prn.1	prn.2	prn.3	prn.4	prn.5	prn.6	prn.7	prn.8	prn.9
5	4	3	2	1	0	1	2	3

In dit geval is het even "kostbaar" om één bus twee perrons van zijn referentieperron af te plaatsen, als twee bussen elk één perron.

Indien men dit een ongewenst effect acht, en men liever één rit vrij ver van zijn referentieperron af plaatst dan twee ritten een beetje, dan kan men ook een iets andere kostenfunctie hanteren, die voor *Rit1* in de volgende reeks resulteert:

prn.1	prn.2	prn.3	prn.4	prn.5	prn.6	prn.7	prn.8	prn.9
15	14	13	12	11	0	11	12	13

Wanneer nu een rit twee perrons van zijn voorkeursperron af geplaatst wordt, levert dat 12 strafpunten op, en het plaatsen van twee bussen elk één perron van hun voorkeursperron, levert 22 strafpunten op. Dus deze laatste kostenfunctie plaatst zo "min mogelijk" ritten af van hun voorkeursperron.

Ook is het mogelijk om de waarde van de status mee te nemen in de kostenfunctie, waarmee men kan uitdrukken dat het de voorkeur verdient om ritten met een "lage" status op een ander perron te plaatsen, dan ritten met een "hoge". Stel de waarde van de status van *Rit1* is 1, en die van *Rit2* is 2. (*Rit2* is bijv. al vooraangekondigd) Dan kunnen we in de bovenstaande berekening de verkregen afstand nog vermenigvuldigen met deze waarde, waarna we op de volgende kosten uitkomen:

	prn.1	prn.2	prn.3	prn.4	prn.5	prn.6	prn.7	prn.8	prn.9
Rit1	15	14	13	12	11	0	11	12	13
Rit2	30	28	22	0	22	24	28	30	32

Op grond waarvan *Rit1* verplaatst zal worden, en niet *Rit2*.

In Sectie 5 wordt een voorbeeld van deze lineaire ongelijkheden en de kostenfunctie gegeven.

2.6 De Status

A.h.v. de status van de reservering wordt bepaald hoe elke rit van invloed is op de kostenfunctie, zoals behandeld is in de vorige sectie.

In het derde punt van Sectie 2.2 is al uitgelegd welke waarden de status van een rit aan zou kunnen nemen, en hoe de status in de loop van de tijd aangepast kan worden na vooraankondiging, vertraging, binnenkomst, etc.

Aan de hand van de status kan ook worden uitgedrukt of een reservering van een rit eigenlijk wel verplaatst kan worden. Bij het construeren van een volgende optimale toewijzing is het bijv. mogelijk om af te dwingen dat alle ritten met een status *op-perron* niet verplaatst zullen worden.

De vraag of een rit met status *vooraangekondigd* in principe verplaatst kan worden is subtieler. Uiteraard is het niet wenselijk, er kunnen zich echter uitzonderingssituaties voordoen, bijv. een vooraangekondigde rit die alsnog veel later binnenkomt, waarbij het wel degelijk zinnig kan zijn om deze rit toch te verplaatsen. Het alternatief is namelijk dat er dan überhaupt geen volgende optimale toewijzing is, wat uiteraard nog minder wenselijk is.

Het lijkt dus voor de hand te liggen om aan de status *vooraangekondigd* een hoge waarde toe te kennen, zodat het "hoge" kosten met zich meebrengt om zo'n rit te verplaatsen. Hierdoor zal zo'n rit pas verplaatst als er echt geen ander alternatief is.

2.7 De detecties

Het aanpassen van de status

Een detectie van een rit leidt tot een aanpassing van de status van de betreffende rit. Er zijn drie soorten detecties af te handelen:

- Detectie op een toevoerweg T, dan wordt de rit "vooraangekondigd" en wordt de status veranderd in *vooraangekondigd-T*.
- Detectie bij een ingang van het station, als een rit gedetecteerd wordt bij ingang I, dan wordt zijn status veranderd in *gearriveerd-I*.
- Detectie bij een vertrek, als een rit gedetecteerd wordt bij uitgang U, dan wordt zijn status veranderd in *vertrokken-U*.

Het doorvoeren van vertragingen

Als een reservering van een rit begint om 10.02, en de rit rijdt een route met het detectiepunt van de vooraankondiging op twee minuten afstand van de ingang, dan wordt de reservering als vertraagd beschouwd als er om 10.00.00 (10.00 en 0 seconden) nog geen detectie heeft plaatsgevonden. Vervolgens wordt de reservering verlengd; als er één keer per minuut bepaald wordt welke ritten vertraagd zijn, dan ligt het voor de hand om de reserveringen, na het constateren van de vertraging, ook met 1 minuut te verlengen.

2.8 De instructies

Als een rit gedetecteerd is bij een toevoerweg, dan zal er bepaald moeten worden of hij gelijk door kan rijden naar zijn toegewezen perron, dan wel zal moeten plaats nemen in een buffer.

Doorrijden naar toegewezen perron of plaatsnemen in buffer?

Op het moment dat een rit bij een ingang wordt gedetecteerd, krijgt de rit de instructie om door te rijden naar het toegewezen perron als er op dat perron plaats is.[1] Anders, dan krijgt de rit de instructie om plaats te nemen in een buffer, hiertoe wordt er dan eerst een willekeurige vrije buffer gekozen.

Aanpassen van instructies

Als een rit in een buffer staat en er komt een plaats vrij op het perron waarop de rit is toegewezen, dan moet de rit een instructie krijgen om op te rijden naar het toegewezen perron. Echter, als dat perron nog bezet is, door een rit die later vertrekt dan gepland, dan kan de rit uiteraard niet op het perron plaatsnemen, en wordt hij als vertraagd beschouwd, op grond waarvan zijn reservering wordt verlengd.

Een andere oplossing zou zijn, dat er dan zo mogelijk alsnog een ander perron gezocht wordt.

2.9 De looptijd en het gebruik van een horizon

De toewijzing bevat al de ritten die binnen een zekere *looptijd* vallen, bijv. een dag.

Om ritten die relatief binnen korte tijd aankomen een iets hogere prioriteit te geven dan ritten die later aankomen kan er een tweetal waarden voor de status geïntroduceerd worden voor *gereserveerde* maar vooralsnog ongedetecteerde ritten; bijv. de waarden *gereserveerd1* en *gereserveerd2*.

Het feit dat een rit "binnen korte tijd aankomt" wordt geformaliseerd door een zgn. *horizon*, van bijv. 15 minuten. Alle ritten die tussen het huidige tijdstip en deze horizon vallen krijgen minstens (merk op dat na detecties of vertragingen de status ook aangepast wordt) de status *gereserveerd2*, en alle "latere" ritten krijgen de status *gereserveerd1*. Elke keer als er een minuut verstreken is, dan moet er worden bepaald welke "nieuwe" ritten binnen de horizon te vallen.

[1] Het zou ook vereist kunnen worden dat als de reservering nog niet is ingegaan, de rit eerst zolang in een buffer moet wachten.

3 Uitbreidingen op het kern–model

Het bovengenoemde kern–model bevat, in onze ogen, de meest essentiële onderdelen van een compact dynamisch busstation, echter voor een concreet busstation zal mogelijk blijken dat niet alle aspecten ervan in dit model te formuleren zijn.

In deze sectie beschrijven we een aantal uitbreidingen op het kern–model, dat voor praktijk voorbeelden relevant kan zijn.

3.1 Het bijvoorbaat uitsluiten van bepaalde toewijzingen

Het is denkbaar dat men wil aangeven dat bepaalde dienstwagen-typen, bepaalde ritten etc. bepaalde perrons niet mogen aandoen. Ook is het mogelijk om aan te geven dat een bepaalde route niet samengaat met een bepaald perron; op deze wijze kan bijv. uitgedrukt worden dat een bus die volgens zijn route het station bij een bepaalde uitgang verlaat, niet op een perron geplaatst kan worden waarvandaan deze uitgang onbereikbaar is.

3.2 Een reserveperron

Er kan een apart perron, het *reserveperron*, gereserveerd worden voor uitzonderingsgevallen. Ritten worden slechts op het reserveperron geplaatst als er geen enkel ander mogelijk perron vrij is; dit wordt in de kostenfunctie afgedwongen door hoge kosten te rekenen voor het plaatsen van een rit op het reserveperron.

3.3 Overstap–aspecten; gecorreleerde ritten

In verband met overstappen is het mogelijk om de afstand tussen de toegewezen perrons van twee ritten zo klein mogelijk houden. Dit kan men verkrijgen door het introduceren van een zgn. "correlatie-factor" tussen twee ritten. Ritten die niet of nauwelijks gecorreleerd zijn hebben tesamen een correlatie-factor 0, ritten die enigszins gecorreleerd zijn (gemiddeld aantal overstappende passagiers) hebben een factor 1, en sterk gerelateerde ritten (veel overstappende passagiers) hebben een factor 2.

In de kostenfunctie wordt dan voor elk paar ritten hun onderlinge afstand vermenigvuldigd met hun correlatie-factor. Om het effect te versterken kan ook het kwadraat van de onderlinge afstand genomen worden.

3.4 Indraai– en uitdraai–ruimte

In sommige gevallen is het denkbaar dat als een bus met een bepaalde lengte een bepaald perron moet opdraaien, er een bepaalde vrije ruimte, de zgn. "indraai–ruimte", vereist wordt op een ander perron. Bijv. als op perron 10 een lange bus wil indraaien, dan met er minstens 5 meter vrije ruimte zijn op perron 9. Dit vergt een minimale uitbreiding van de datastructuren uit het kern-model. Op dezelfde wijze kan er uitgedrukt worden dat bij het vertrekken van een perron er een vrije ruimte, de zgn. "uitdraai-ruimte", is vereist op een ander perron.

3.5 Het afzwakken van condities

Als een toewijzing inconsistent is, en er een nieuwe optimale en consistente toewijzing opgesteld moet worden, dan is het in theorie mogelijk dat de condities geen enkele andere consistente toewij-

zing toelaten. Mocht dat inderdaad optreden, dan kan het voorkomen worden door de condities af te zwakken.

Zo werd er tot nu toe vereist dat de totale lengte van de bussen die tegelijkertijd op een perron staan, niet de lengte van het perron mag overschrijden. Dit kan ook enigszins afgezwakt worden, en wel door het geven van strafpunten voor de mate waarin de lengte overschreden wordt, of zelfs een hoog veelvoud ervan.

Een consequentie hiervan is dat er vaker een nieuwe toewijzing mogelijk is; immers, de kostenfunctie zal afdwingen dat eerst de toewijzingen zonder perron–overschrijdingen aan bod komen, pas als al deze toewijzingen afvallen (om wat voor redenen dan ook) dan zal de meest optimale met een perron–overschrijding bepaald worden. In het kern–model zou in dit laatste geval geen volgende toewijzing mogelijk zijn.

De prijs is echter wel dat het vinden van zo'n volgende toewijzing, na deze uitbreiding van de kostenfunctie, meer tijd zal kosten.

4 Het prototype

Het kern-model uit Sectie 2 en uitbreidingen uit de Secties 3.1 en 3.2 zijn uitgewerkt in een eerste prototype, dat enigszins is gebaseerd op de gegevens van Station Apeldoorn [DOWA94].

Dit prototype bestaat uit een aantal tools dat aangestuurd wordt vanuit Toolbus-processen [BK95]. Deze Toolbus-processen wisselen d.m.v. communicatie onderling gegevens uit. Behalve tijdens de initialisatiefase gebeuren in dit prototype alle communicaties *asynchroon*, wat inhoudt dat het ene proces niet op het andere hoeft te wachten, maar dat de gecommuniceerde data in een buffer worden opgeslagen totdat het ontvangende proces ze opneemt.

4.1 De cyclus van de *Controller*

Het centrale Toolbus-proces is de *Controller* dat alle andere processen instrueert. Dit proces doorloopt de onderstaande cyclus, waarbij het *vorige* tijdstip het tijdstip is waarop de cyclus de vorige keer werd doorlopen. Deze cyclus zal in het vervolg in meer detail uitgelegd worden, waarbij verwezen zal worden naar de nummers van de onderdelen waaruit de cyclus bestaat.

start cyclus

1. Bepaal het huidige tijdstip.
2. Geef de meest recente detecties (d.w.z. alle detecties tussen het huidige en het vorige tijdstip.)
3. Verwerk deze detecties en bepaal daarbij welke ritten evt. naar een buffer of perron moeten oprijden. (Merk op dat de toewijzing na deze aanpassing nog steeds consistent zal zijn, omdat er immers alleen de status maar niet de lengte van de reservering van ritten aangepast wordt.)
4. Bepaal welke ritten moeten vertrekken.
5. Geef de instructies door.
6. Als het één minuut geleden is, of meer, voer dan de volgende wijzigingen door:

   start doorvoeren wijzigingen.

   (a) Bepaal de vertraagde ritten van de afgelopen periode (de afgelopen minuut) en verleng hun reserveringen.

   (b) Schuif de horizon op. (Dit zal hierna worden uitgelegd.)

   (c) Stel een perrontoewijzing op, en maak deze consistent. (De huidige toewijzing is immers, na het bepalen van de vertragingen en het verschuiven van de horizon, mogelijkerwijs inconsistent geworden! In theorie is het mogelijk dat er geen volgende consistente perrontoewijzing opgesteld kan worden, in het prototype is dit geval niet verder afgehandeld)

   (d) Verwerk de voorgestelde wijzigingen.

   einde doorvoeren wijzigingen.
7. Bepaal de huidige haltetoewijzing. (Als er tenminste een consistente toewijzing is.)
8. Druk deze haltetoewijzing af op het user-interface.

einde cyclus

4.2 Het tijdstip en de frequentie van het doorvoeren van de wijzigingen

In de bovenstaande cyclus worden de wijzigingen in principe om de minuut doorgevoerd. Deze *verwerk-wijziging-termijn* zou ook een halve minuut kunnen zijn, of elk willekeurig interval. De keus van één minuut komt overeen met de dienstregeling, die immers ook op de minuut nauwkeurig geformuleerd is.

Hoe kleiner deze verwerk-wijziging-termijn gekozen wordt, hoe vaker er een optimale toewijzing berekend zal worden, wat een relatief dure operatie is die enige tijd (enkele seconden) kan duren.

Als de verwerk-wijziging-termijn relatief klein wordt gekozen (bijv. 15 sec.), dan kan het gebeuren dat het doorvoeren van de wijzigingen langer duurt (bijv. 20 sec.). In dat geval zal de volgende keer dat de cyclus doorlopen wordt eerst de detecties verwerken die gedurende deze 20 seconden zijn opgetreden, waarna deze bijbehorende wijzigingen direct verwerkt zullen worden. Voor de correctheid van het protocol maakt het echter niet uit dat de verwerk-wijziging-termijn overschreden is.

Het is dus ook mogelijk dat deze termijn op nul gezet wordt. In dat geval worden elke keer wanneer de cyclus doorlopen wordt de wijzigingen doorgevoerd.

4.3 Het verwerken van detecties

In de bovenstaande cyclus wordt een detectie over het algemeen iets later, enkele seconden, verwerkt, dan het moment waarop deze optreedt. Voor bepaalde detecties, zoals vooraankondigingen, is dit tijdsverschil (mits slechts een aantal seconden, en geen minuten) geen punt. Echter, voor een rit die op het station aankomt is het wel van belang. Immers, als een rit binnenkomt dan kan het enkele seconden duren voordat de detectie verwerkt wordt, en de instructie om op een buffer plaats te nemen, of om op te rijden naar een perron wordt afgebeeld.

In het prototype wordt hier verder niet op ingegaan; er wordt impliciet van uitgegaan dat een binnenkomende rit enige tijd kan wachten voor de bijbehorende instructie wordt afgebeeld.

4.4 Varianten op de cyclus

Er zijn ook andere varianten van de bovenstaande cyclus mogelijk. Zo zou het doorvoeren van de wijzigingen kunnen worden "uitbesteed" aan een apart hulpproces, zodat het verwerken van de detecties niet opgehouden wordt.

Een complicatie hierbij is dat dit hulpproces tot de conclusie kan komen dat een bepaalde rit aan een ander perron moet worden toegewezen, terwijl diezelfde rit al kan zijn binnengekomen en al kan hebben plaatsgenomen op het aan hem toegewezen perron. In dat geval kan de wijziging dus niet worden doorgevoerd, en moet het hulpproces opnieuw een consistente perrontoewijziging opstellen. Overigens wordt deze situatie in de huidige kostenfunctie vrijwel uitgesloten, omdat een vooraangekondigde rit alleen tegen hoge kosten verplaatst kan worden.

4.5 Het gebruik van horizons

Tijdens de looptijd wordt niet de gehele toewijzing over deze looptijd gemanipuleerd en afgebeeld. Uitgaande van het huidige tijdstip en een horizon wordt er telkens een gedeelte geselecteerd.

In het Toolbus-script is er sprake van een `KorteHorizon`, dit geeft de horizon aan waarbinnen de ritten meegenomen worden in de optimalisatie. Deze horizon wordt in overeenstemming gekozen met de tijd die de ritten nodig hebben om vanaf de "buitenste" detectoren naar het station te rijden, en de rekencapaciteit van het systeem. In het huidige prototype is deze horizon 15 minuten.

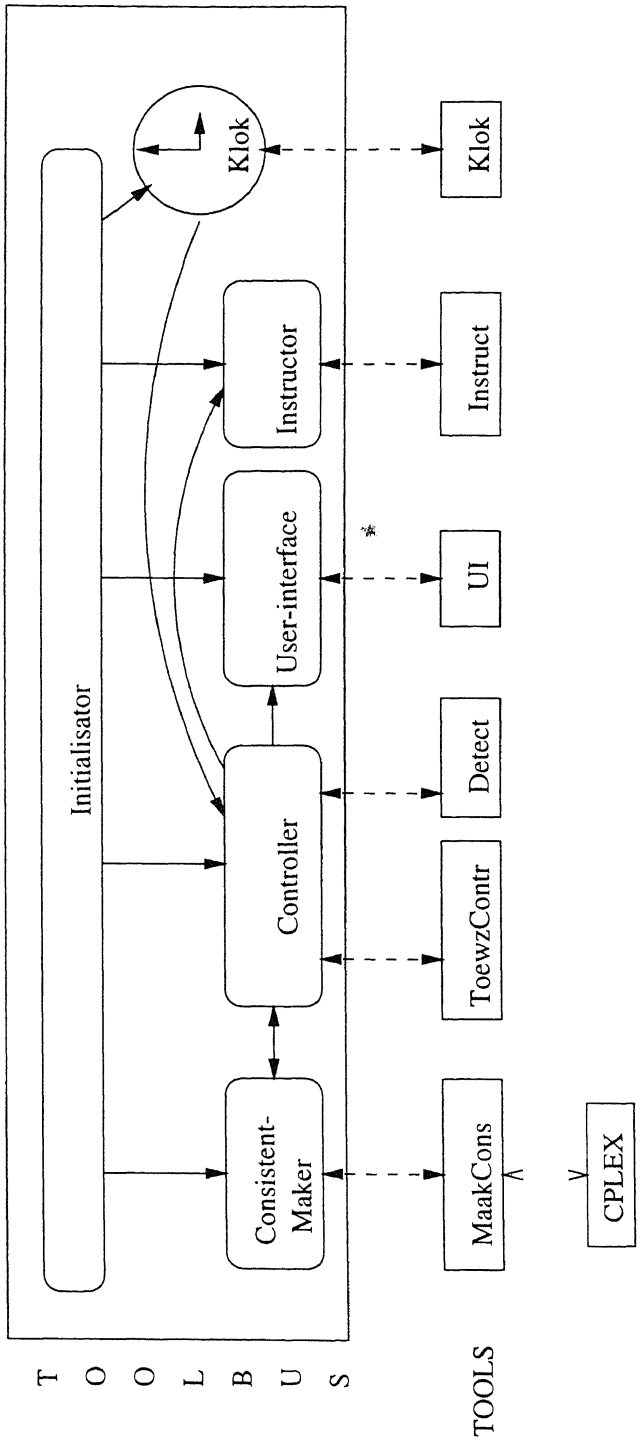

Figure 2: Grafisch overzicht van de Toolbus processen en hun tools

Vervolgens is er in het Toolbus-script ook sprake van een `LangeHorizon`, dit geeft de horizon aan waarbinnen de ritten worden afgebeeld op het user-interface. Deze horizon is bijv. 60 minuten.

4.6 Een overzicht van de Toolbus-processen en hun tools

Naast de *Controller* is er nog een aantal andere Toolbus-processen, elk met hun eigen tools, zoals geschetst in figuur 2. [2] Hieronder worden deze processen en tools kort aangegeven, in Sectie 4.8 zal het gehele Toolbus-script behandeld worden en in de Appendix zullen de tools verder worden behandeld.

- De *Initialisator* start de tools en voorziet deze van de benodigde start-waarden. Vervolgens worden de overige Toolbus-processen gestart.
- De *Controller* voert na initialisatie de cyclus uit die in de vorige sectie is gegeven. Dit proces stuurt de volgende tools aan:
  - `ToewzContr` (geschreven in Perl, zie [WS90]). Dit tool houdt de complete toewijzing bij. Hierin worden de wijzigingen doorgevoerd, en op verzoek van de Controller levert het een perron- of een haltetoewijzing op.

    Het wordt door de *Initialisator* geïnitialiseerd met het starttijdstip en het stoptijdstip van gehele looptijd. Vervolgens leest het van een file de benodigde informatie in om de gehele initiële toewijzing over deze looptijd op te stellen. Deze file komt overeen met de dienstregeling waarin wordt vastgelegd op elke minuten in het uur een lijn aankomt en vertrekt, evenals het bijbehorende (voorkeurs)perron.

    Ook krijgt het een parameter mee die de `KorteHorizon` aangeeft, zodat het weet welke ritten de status `gereserveerd1`, dan wel `gereserveerd2` moeten krijgen.

    Tijdens het opstellen van de initiële toewijzing wordt, d.m.v. een random-generator, ook bepaalt hoe laat een rit daadwerkelijk vooraangekondigd wordt resp. aankomt op het station.

    Het is echter op voorhand niet bekend hoelaat een rit van een perron zal vertrekken, dit hangt immers niet alleen af van de reeds gegenereerde aankomsttijd, maar zal ook dynamisch bepaald worden door de dan geldende toewijzing en de werkelijke aankomsttijden van de voorgaande ritten.

    Al deze detectie-gegevens worden weer teruggegeven aan de Toolbus, zodat deze de detectie-gegevens weer kan doorgeven aan het tool `Detect`.
  - `Detect` (geschreven in TCL/TK, [Ous94]). Dit tool wordt geïnitialiseerd met de hiervoor genoemde detectie-gegevens zoals opgesteld door het tool `ToewzContr`.

    Het kan worden gevraagd om alle detecties aan de Controller af te geven die plaats hebben gehad tussen een tweetal tijdstippen. Deze detecties worden dan ook afgebeeld op het scherm.
- De *Consistent-maker*, dit tool krijgt een perrontoewijzing binnen en bepaalt, uitgaande van deze toewijzing, de meest optimale toewijzing.

[2] In dit plaatje zijn de communicaties tussen de Initialisator en de tools niet met pijlen weergegeven, evenals de communicaties om acknowledgements aan de Controller terug te geven.

Het tool maakt zelf weer gebruik van het tool Cplex, een implementatie van een lineair programmerings-algoritme (zie tekstboeken als[Wil90] of [Sha90], of de manual van Cplex [CPL]) .

- Een *Klok*, die met een bepaalde frequentie, bijv. om de 15 sec. een puls geeft. De timer zelf wordt in de Toolbus bijgehouden, na elke puls wordt het nieuwe tijdstip door een bijbehorend `klok`-tool afgebeeld.
- De *User-interface*, die de haltetoewijzing tot aan een bepaalde horizon (`lange horizon`) afbeeldt. Ook wordt per perron de informatie gegeven over de ritten die eraan zijn toegewezen.
- Een *Instructor*, die de instructies afbeeldt.

In de volgende sectie zal het bijbehorende Toolbus-script in detail gegeven worden. De codes van de tools worden hier verder niet gegeven.

4.7 Enkele voorbeelden van Toolbus scripts

Voordat in de volgende sectie het Toolbus-script van het CDB gepresenteerd wordt, zal in deze sectie eerst de Toolbus uitgelegd worden a.h.v. een aantal kleine en eenvoudige scripts. Voor een grondiger uitleg verwijzen wij naar [BK95].

Een enkel Toolbus-proces met een tool

Een Toolbus-proces kan communiceren met één of meerdere tools. Stel er is een tool met de naam `Suc` die d.m.v. de functie `suc` de successor van een integer `N` uit kan rekenen (`suc(n)=n+1`), dan kan deze functie vanuit een Toolbus script als volgt worden aangeroepen:

```
snd-eval(Suc,suc(N))
```

m.a.w., de Toolbus stuurt aan het tool `Suc` het verzoek om de term `suc(N)` te evalueren. Vervolgens stuurt het tool een waarde terug, die m.b.v.

```
rec-value(Suc,N?)
```

door de Toolbus in de variabele `N` wordt ingelezen.

Voordat er echter een dergelijke communicatie kan plaatsvinden met het tool, moet het eerst bekend gemaakt, en gestart, worden.

De definitie gebeurt a.h.v.:

```
tool successor is {command = "perl-adapter -script perl.suc"}
```

dat aan het tool `succesor` het shell script `perl.suc` verbindt. Dit script is geschreven in de taal Perl, en de `perl-adapter` verzorgt de communicatie tussen de Toolbus en dit tool. Vervolgens wordt met het commando

```
execute(successor,Suc?)
```

een instantie van dit tool gestart, de Toolbus geeft deze instantie een bepaalde identificatie die wordt ingelezen in de variabele `Suc`. `Suc` is gedeclareerd als variable van het type `successor`; een toolnaam wordt ook beschouwd als het type van alle mogelijke identifiers van instanties van dit tool.

Uiteindelijk wordt het proces P gestart met de regel

```
toolbus(P)
```

De sequentiële compositie wordt in Toolbus scripts aangeduid met de punt (.). Het complete script is dan als volgt:

```
process P                          %% Declaratie Toolbus processen,
let Suc:successor,                 %% in dit voorbeeld alleen proces P
    N:int
in
 execute(successor,Suc?).
 N := 0.
 snd-eval(Suc,suc(N)).
 rec-value(N?).
end

tool succesor is {command = "perl-adapter -script perl.suc"}
                                   %% Declaratie externe tools
toolbus(P)                         %% Start van de Toolbus processen
```

De operatoren in een proces

Toolbus statements kunnen op verschillende wijzen gecombineerd worden, in het bovenstaande voorbeeld hebben we de *sequentiele compositie* al gezien; `p.q` voert eerst `p` uit en dan `q`. Andere voorbeelden zijn:

alternatieve compositie	`p+q`
herhaling	`p*q`
conditionele keuze	`if B then p else q`

`p+q` executeert `p` *óf* `q` (één van beide!), `p*q` executeert `p` nul of meer keren, en dan `q`, `if B then p else q` executeert `p` als de boolean `B` waar is, anders executeert het `q`. Vervolgens is er het atomaire script `delta`, dat nooit uitgevoerd zal worden. Het wordt bijv. gebruikt in de context `p*delta`, dat `p` oneindig vaak zal uitvoeren (totdat de gehele Toolbus afgebroken wordt).

Parallele processen

Een Toolbus-script kan bestaan uit meerdere processen die parallel aan elkaar lopen. Dit houdt in dat de Toolbus van elk van deze processen een eerstvolgende commando kan uitvoeren; er is hierbij in principe op voorhand geen volgorde bepaald.

Neem bijvoorbeeld het volgende script:

```
process P                          process Q
let X:int                          let Y:int
in                                 in
 X := 0.                            Y := 1.
end                                end

toolbus(P,Q)
```

Dit script kent twee mogelijke executie-paden: eerst de `X := 0` binnen `P` en dan de `Y := 1` binnen `Q`, of andersom. De implementatie van de Toolbus zal één van beide executie-paden kiezen.

Synchrone communicatie tussen processen

Toolbus processen kunnen op verschillende wijzen data met elkaar uitwisselen; t.w. *synchroon* en via *broadcasting*.

Een synchrone communicatie is binair; er zijn precies twee partijen, een zender en een ontvanger. Beide processen wachten tot het andere proces ook zover is, zodra dit het geval is, vindt de communicatie plaats, en wordt de data "overhandigd". Synchrone communicatie kent twee complementaire acties, `snd-msg(code0,Data)` (*send message*, de zendende actie) en `rec-msg(code1,V?)` (*receive message*, de ontvangende actie), als `code0` en `code1` overeenkomen, dan zal `V` de waarde van `Data` krijgen. In het script

```
process Zender is snd-msg(boodschap1,1)

process Ontvanger is
let N:int in
 rec-msg(boodschap1,N?). p
+rec-msg(boodschap2,N?). q
endlet

toolbus(Zender,Ontvanger)
```

komt de `snd-msg(boodschap1,1)` van de `Zender` overeen met de `rec-msg(boodschap1,N?)` van de `Ontvanger`, waardoor de communicatie plaats zal vinden en `N` de waarde 1 krijgt, waarna de `Ontvanger` met `p` vervolgt. In het volgende script

```
process Zender is snd-msg(boodschap,1)

process Ontvanger is
let N:int in
rec-msg(boodschap,N?) delay(10)
endlet

toolbus(Zender,Ontvanger)
```

is de `Zender` direct klaar voor de communicatie, de `Ontvanger` wacht echter eerst 10 seconden, d.m.v. het `delay`-statement. De `Zender` zal dus ook 10 seconden moeten wachten, waarna de communicatie plaats zal vinden, en `N` de waarde 1 verkrijgt.

Asynchrone communicatie tussen processen via *broadcasting*

Een proces kan echter ook een boodschap *rondsturen* (broadcasten) aan alle andere processen, waarbij het niet hoeft te wachten totdat een ander proces de boodschap heeft kunnen ontvangen. Als er sprake is van slechts één ontvanger dan wordt het ook wel *asynchrone communicatie* genoemd, omdat het moment van versturen niet samenvalt (*asynchroon* is) met het moment van ontvangen.

Als een ander proces zo'n boodschap zou willen ontvangen, moet het zich er eerst op *abonneren* (d.m.v. `subscribe`).

```
process Zender is snd-note(boodschap(1))

process Ontvanger is
```

```
let N:int in
subscribe(boodschap(<int>)).
rec-note(boodschap(N?)) delay(10)
endlet

toolbus(Zender,Ontvanger)
```

In dit voorbeeld zal de `Zender` de boodschap `1` direct rondzenden, en na 10 seconden zal de `Ontvanger` deze boodschap ontvangen. In tussentijd wordt de boodschap bijgehouden in een buffer. Deze buffer werkt als een *first-in-first-out*-queue, dat wil zeggen dat de ontvanger de verschillende boodschappen ontvangt in de volgorde waarin ze zijn verstuurrd.

4.8 Het Toolbus script

In deze sectie geven we het gehele Toolbus-script van het prototype, met enige informele uitleg. Alleen enkele kleine hulpprocessen zijn achterwege gelaten. Tijdstippen, als 7.10 en 32 seconden, worden in de Toolbus gepresenteerd als de lijst (type `list`) van de vorm `[7,10,32]`.

Datastructuren als perrontoewijzingen, haltetoewijzingen e.d., worden in de Toolbus gerepresenteerd als strings (type `str`); alleen de aangesloten tools zullen deze strings verder interpreteren.

Het *initialisatie* proces `CDB`

```
process CDB(
   StartTyd       :list, # Begintijdstip looptijd
   StopTyd        :list, # Eindtijdstip looptijd
   Interval       :int,  # Stapgrootte van de klok in minuten
   KorteHorizon   :int,  # Horizon waarbinnen geoptimaliseerd wordt
   LangeHorizon   :int   # Horizon die afgebeeld wordt
) is
let Klok          :klok,
    ToewzContr    :toewz,
    Detect        :detect,
    MaakCons      :maakcons,
    Instruct      :instruct,
    Ui            :ui,

    N             :int,
    Stap          :bool,
    AlleDetecties :str,
    HalteToewz    :str,
    Ack           :str
in
```

Bij het initialiseren van de klok wordt aan de gebruiker gevraagd of elke tijdstap door de klok zelf genomen moet worden, dan wel expliciet moet worden aangegeven door de gebruiker, door een knop op een button. In het laatste geval levert `snd-eval(Klok,init)` de waarde `1` af, waarna `Stap` op `true` wordt gezet.

```
execute(klok,Klok?) .
snd-eval(Klok,init) .
  rec-value(Klok,N?) .
Stap := equal(N,1) .
```

ToewzContr is het tool dat de complete toewijzing voor de gehele looptijd opstelt en bijhoudt. Het opstellen van de initiële toewijzing wordt gedaan door onderstaande aanroep `init(..,..,..)`. De parameters StartTyd en StopTyd geven het begin resp. het einde aan van de looptijd. Alle ritten tussen StartTyd en KorteHorizon worden geïnitialiseerd met status gereserveerd2, alle overige ritten met gereserveerd1. De functie `init` wordt behandeld in Sectie B.1.1.

```
execute(toewz,ToewzContr?) .
snd-eval(ToewzContr,init(StartTyd,StopTyd,KorteHorizon)) .
 rec-value(ToewzContr,AlleDetecties?) .
```

MaakCons is het tool dat de perrontoewijzing na wijzigingen weer consistent maakt.

```
execute(maakcons,MaakCons?) .
```

Detect is het tool dat de detecties bijhoudt en afgeeft. Het wordt geïnitialiseerd met de lijst van alle detecties, zoals opgesteld door ToewzContr, de waarde van de acknowledgement Ack is verder niet meer van belang.

```
execute(detect,Detect?) .
snd-eval(Detect,init(AlleDetecties)) .
 rec-value(Detect,Ack?) .
```

Instruct is het tool dat de instructies afbeeldt.

```
execute(instruct,Instruct?).
```

Ui is het user-interface tool dat de meest recente haltetoewijzing afbeeldt, en tevens de informatie per perron.

```
execute(ui,Ui?) .
snd-eval(Ui,init(StartTyd)).
 rec-value(Ui,Ack?).
```

Vervolgens wordt aan ToewzContr de initiële haltetoewijzing opgevraagd, die vervolgens wordt meegegeven aan het tool Ui.

```
snd-eval(ToewzContr,stelHalteToewzOp(StartTyd,LangeHorizon)).
 rec-value(ToewzContr,HalteToewz?).
snd-eval(Ui,nwScherm(StartTyd,HalteToewz,"Initiele Toewyzing")).
 rec-value(Ui,Ack?).
```

Nu alle tools geïnitialiseerd zijn, worden vervolgens de overige processen in de Toolbus geïnitialiseerd. Elk proces krijgt de identificaties van de benodigde tools mee, evenals bepaalde startdata. Als een proces daadwerkelijk gestart is, dan antwoord het met de boodschap ok.

```
 snd-msg(klok,Klok,StartTyd,StopTyd,Interval,Stap).
  rec-msg(klok,ok).
 snd-msg(controller,ToewzContr,Detect,KorteHorizon,LangeHorizon).
  rec-msg(controller,ok).
 snd-msg(maakcons,MaakCons).
  rec-msg(maakcons,ok).
 snd-msg(instruct,Instruct).
  rec-msg(instruct,ok).
 snd-msg(ui,Ui).
  rec-msg(ui,ok)
endlet
```

Het Controller-proces

Het Controller-proces is de spil van het gehele regulerings-systeem.

Het proces bestaat uit een grote cyclus, zoals hiervoor beschreven in Sectie 4.1, die gedurende de hele looptijd herhaald wordt. Het begin van de cyclus bestaat uit het opvragen van de detecties, en het verwerken ervan. Het doorvoeren van de wijzigingen wordt niet elke keer uitgevoerd, hoogstens één keer per minuut, zie Sectie 4.2.

```
process CONTROLLER is
let ToewzContr   : toewz,
    Detect       : detect,
    Vorig        : list,  %% Vorige keer dat loop werd doorlopen
    VorigUpd     : list,  %% Vorige keer dat subloop werd doorlopen
    Nu           : list,  %% huidig tijdstip
    KorteHorizon : int,
    LangeHorizon : int,
    HalteToewz   : str,
    PerronToewz  : str,
    VoorAank     : str,   %% Gedetecteerde vooraankondigingen
    StatAank     : str,   %% Gedetecteerde aankomsten op station
    PerVertr     : str,   %% Gedetecteerde vertrekken van perron
    Vertraagd    : str,   %% Vertraagde ritten
    Nieuw        : str,   %% Nieuwe ritten, aan einde horizon
    NaarPerron   : str,   %% Oprij-instructies, naar perron
    NaarBuffer   : str,   %% Oprij-instructies, naar buffer
    VanPerron    : str,   %% Vertrek-instructies, van perron
    VerwVanPerron: str,   %% Ritten en hun verwachte vertrektijdstippen
    Wyzigingen   : str,   %% Voorgestelde wijzigingen om toewijzing weer
                          %% consistent te maken
    Update       : bool,
    Ack          : str
in
```

Nog voor de initialisatie abonneert het proces zich op de boodschappen `tyd(<list>)` (verstuurt door het proces `KLOK`), en `wyzigingen(<str>)` (verstuurt door het proces `MAAKCONS`.

De variabelen `Vorig` en `Nu` worden hier geïnitialiseerd op `[7,0,0]` (7 uur, precies), en `VorigUpd` op een eerder tijdstip (10 voor 7).

Dan wacht het op de instructie `controller` (verstuurd door het initialisatie proces CDB) waarna de initialisatie kan worden afgerond, met het terugsturen van ok, en de cyclus kan beginnen.

```
subscribe(tyd(<list>)) .
subscribe(wyzigingen(<str>)) .
Vorig := [7,0,0] .
VorigUpd := [6,50,0].
Nu    := [7,0,0] .
rec-msg(controller,ToewzContr?,Detect?,KorteHorizon?,LangeHorizon?) .
 snd-msg(controller,ok) .
(%% begin loop
```

Eerst haalt het de meest recente tijd op (stap 1 uit de cyclus van Sectie 4.1), daarvoor wordt de tyd-queue doorlopen totdat deze leeg is (`no-note(tyd)`).

```
rec-note(tyd(Nu?)).(rec-note(tyd(Nu?))*no-note(tyd)) .
```

Dan (stap 2 uit de cyclus van Sectie 4.1) verkrijgt het de meest recente detecties van `Detect`, gesplitst over `VoorAank`, `StatAank` en `PerVertr`.

```
snd-eval(Detect,geefDetecties(Vorig,Nu)) .
 rec-value(Detect,detecties(VoorAank?,StatAank?,PerVertr?)) .
```

Vervolgens worden de detecties verwerkt in de toewijzing, stap 3 uit de cyclus van Sectie 4.1. Dit houdt in dat de status van elke zojuist gedetecteerde rit wordt aangepast, zie Sectie 2.7. De lengtes van de reserveringen e.d. blijven ongewijzigd; het verwerken van detecties kan een consistente toewijzing, met de huidige kostenfunctie en condities, dus nooit inconsistent maken. Merk op dat dit niet meer op gaat als er condities toegevoegd zouden worden van de vorm "er mogen niet meer dan twee vooraangekondigde ritten op een perron geplaatst worden".

Als resultaat geeft dit de oprijd-instructies naar perrons en buffers. Van de ritten die het perron oprijden worden de verwachte vertrektijden opgesteld, deze informatie wordt afgeleverd in `VerwVanPerron`. A.h.v. deze laatste informatie kan het `Detect` tool dan de feitelijke vertrekmomenten opstellen, d.w.z. dat per rit in `VerwVanPerron` het moment wordt bepaald waarop het vertrek gedetecteerd zal worden t.b.v. de simulatie.

```
snd-eval(ToewzContr,verwerkDetecties(Nu,VoorAank,StatAank,PerVertr)).
 rec-value(ToewzContr,opRydInstr(NaarPerron?,NaarBuffer?,VerwVanPerron?)).
snd-eval(Detect,stelFeitVertrOp(VerwVanPerron)).
 rec-value(Detect,Ack?).
```

Dan wordt er bepaald welke ritten er tussen `Vorig` en `Nu` moeten vertrekken, stap 4 uit de cyclus van Sectie 4.1. Het resultaat, `VanPerron`, wordt samen met de oprijd-instructies verstuurd naar het instructie-proces (stap 5 uit de cyclus van Sectie 4.1).

```
snd-eval(ToewzContr,stelVertrekInstrOp(Vorig,Nu)) .
 rec-value(ToewzContr,VanPerron?) .
snd-note(instructies(Nu,NaarPerron,NaarBuffer,VanPerron)).
```

Vervolgens worden die aanpassingen verricht die de consistentie van de toewijzing kunnen aantasten. Omdat het vervolgens weer consistent maken van de toewijzing een "dure" operatie is (die uitgevoerd wordt door het maakcons-proces), worden deze aanpassingen hoogstens één keer

per minuut verricht, zie stap 6 uit de cyclus van Sectie 4.1. De aanroep van het hulp-proces `LessMinuut(VorigUpd,Nu,Update?)` (waarvan de definitie verder niet gegeven wordt in dit document) kent `true` aan `Update` toe als `VorigUpd` meer dan een minuut kleiner is dan `Nu`.

Dan worden de vertragingen bepaald: de reserveringen worden verlengd van al die ritten die tussen `VorigUpd` en `Nu` volgens planning hadden moeten worden gedetecteerd. Ook wordt de horizon opgeschoven, wat inhoudt dat alle ritten met status `gereserveerd1` die aan de `KorteHorizon` verschijnen, een nieuwe status `gereserveerd2` krijgen. Vervolgens wordt de `PerronToewz` opgesteld, waaraan de nieuwe, aan de `KorteHorizon` verschenen, ritten zijn toegevoegd. Deze `PerronToewz` is mogelijkerwijs inconsistent geworden, door het verlengen van de reserveringen of door de toevoeging van nieuwe ritten, en daarom opgestuurd naar het maakcons-proces. Hierbij worden ook de vertraagde en de nieuwe ritten opgestuurd; als `PerronToewz` inconsistent is, dan zullen de ritten uit `Vertraagd` en `Nieuw` de eerste kandidaten zijn om te worden verplaatst.

```
LessMinuut(VorigUpd,Nu,Update?) .
if Update then
 snd-eval(ToewzContr,bepaalVertragingen(VorigUpd,Nu)) .
  rec-value(ToewzContr,Vertraagd?) .
 snd-eval(ToewzContr,schuifHorizonOp(VorigUpd,Nu,KorteHorizon)) .
  rec-value(ToewzContr,Nieuw?) .
 snd-eval(ToewzContr,stelPerronToewzOp(Nu,KorteHorizon)) .
  rec-value(ToewzContr,PerronToewz?).
 snd-note(maakcons(PerronToewz,Vertraagd,Nieuw)).
 VorigUpd := Nu .
```

In deze versie wordt gewacht totdat de noodzakelijke wijzigingen weer teruggestuurd worden. Het is echter ook denkbaar dat direkt (waarbij dus niet gewacht wordt op het verkrijgen van de nieuwe toewijzing) weer teruggegaan wordt naar het begin van de loop, om zodoende de verse detecties gelijk te verwerken. Dit laatste scenario levert wel, in theorie althans, de situatie op waarbij een binnenkomende rit op een perron geplaatst wordt, terwijl even later blijkt dat deze rit van perron had moeten wijzigen. (Echter, gegeven het feit dat ritten in aankomst weliswaar wijzigbaar zijn, maar een hoge status hebben, is dit scenario vrijwel uitgesloten). Deze situatie moet eerst worden uitgewerkt, voordat het scenario wordt toegestaan. Door het parallelisme van dit scenario is het gedrag ook moeilijker te doorgronden, daarom wordt hier als eerste versie een eenvoudiger versie gepresenteerd.

Nadat de `Wyzigingen` worden ontvangen, worden ze verwerkt, en kan er een haltetoewijzing worden opgesteld. (Gemakshalve is er vanuit gegaan dat er altijd een oplossing gevonden is.)

Uiteindelijk wordt deze `HalteToewz` verstuurd naar de user-interface (de stappen 7 en 8 uit de cyclus van Sectie 4.1).

```
rec-note(wyzigingen(Wyzigingen?)) .
snd-eval(ToewzContr,verwerkWyzigingen(Wyzigingen)) .
 rec-value(ToewzContr,Ack?) .
snd-eval(ToewzContr,stelHalteToewzOp(Nu,LangeHorizon)) .
 rec-value(ToewzContr,HalteToewz?).
snd-note(nwScherm(Nu,HalteToewz,"Consistente toewijzing"))
```

Voor het geval dat `Update` de waarde `false` heeft, is er nog een `tau` (die zich gedraagt als een *skip*) noodzakelijk, om de `if Update then .. else .. fi` niet te laten blokkeren.

Dan wordt `Vorig` op `Nu` gezet, en wordt de loop bij het begin hervat.

```
  else tau fi .
  Vorig := Nu
 %% end loop
 ) * delta
endlet
```

Het Maak-consistent-proces

Het Maak-consistent-proces abonneert zich op verzoeken van het Controller-proces om een toewijzing consistent te maken. Bij zo'n verzoek worden ook de vertraagde en de nieuwe ritten meegestuurd, zoals in de vorige sectie al is uitgelegd.

Als een dergelijk verzoek ontvangen wordt, dan wordt het doorgestuurd naar het `MaakCons` tool. Dit tool stelt de benodigde wijzigingen op (als `PerronToewz` al consistent is, dan is `Wyzigingen` leeg). Merk op dat dit wel enige tijd kan kosten, bijv. zo'n 5 à 10 seconden. Vervolgens worden de `Wyzigingen` weer verstuurd aan het Controller-proces.

```
process MAAKCONS is
let MaakCons:maakcons,
    PerronToewz : str,
    RitInfo     : str,
    Vertraagd   : str,
    Nieuw       : str,
    Wyzigingen  : str
in
subscribe(maakcons(<str>,<str>,<str>)).
 rec-msg(maakcons,MaakCons?).
snd-msg(maakcons,ok).
(rec-note(maakcons(PerronToewz?,Vertraagd?,Nieuw?)) .
 snd-eval(MaakCons,maakcons(PerronToewz,Vertraagd,Nieuw)) .
 rec-value(MaakCons,Wyzigingen?) .
 snd-note(wyzigingen(Wyzigingen))
)*delta
endlet
```

Het Instruct-proces

Het Instruct-proces abonneert zich op verzoeken om instructies af te beelden. Zodra zo'n verzoek ontvangen wordt, dan wordt het in z'n geheel doorgestuurd naar het `Instruct`-tool.

```
process INSTRUCT is
let Instruct   : instruct,
    Tyd        : list,
    NaarPerron : str,
    NaarBuffer : str,
    VanPerron  : str,
    Ack        : str
in
subscribe(instructies(<list>,<str>,<str>,<str>)) .
 rec-msg(instruct,Instruct?).
```

```
snd-msg(instruct,ok).
(rec-note(instructies(Tyd?,NaarPerron?,NaarBuffer?,VanPerron?))  .
 snd-eval(Instruct,instructies(Tyd,NaarPerron,NaarBuffer,VanPerron)) .
  rec-value(Instruct,Ack?)
) * delta
endlet
```

Het Userinterface-proces

Het Userinterface-proces abonneert zich op verzoeken om het scherm opnieuw op te stellen. Telkens wordt de gehele queue van verzoeken uitgelezen totdat deze queue leeg is, waarna het meest recente verzoek daadwerkelijk naar het `ui`-tool gestuurd wordt. Omdat er vervolgens gewacht wordt op een acknowledgement dat het tool klaar is, kan het zijn dat de queue zich inmiddels weer gevuld heeft.

```
process UI is
let Ui:ui,
    Tyd : list,
    HalteToewz : str,
    Boodschap :str,
    Ack :str
in
subscribe(nwScherm(<list>,<str>,<str>)) .
 rec-msg(ui,Ui?).
snd-msg(ui,ok).
(rec-note(nwScherm(Tyd?,HalteToewz?,Boodschap?)) .
 (rec-note(nwScherm(Tyd?,HalteToewz?,Boodschap?)) * no-note(nwScherm)).
 snd-eval(Ui,nwScherm(Tyd,HalteToewz,Boodschap)) .
  rec-value(Ui,Ack?)
) * delta
endlet
```

Het Klok-proces

Het Klok-proces verhoogt telkens de `Tyd` met `Interval`, de stap-grootte in seconden. Als `Stap` is `true`, dan wordt er m.b.v. `snd-eval(Klok,stap).rec-value(Klok,Ack?)` gewacht totdat de gebruiker op de stap-button gedrukt heeft. Anders, dan wordt er m.b.v. `tau delay(Interval)` een delay van lengte `Interval` afgedwongen.

Vervolgens wordt de tijd opgehoogd, en begint de loop weer van voor af aan mits de looptijd niet verstreken is. Als dit wel het geval is, dan wordt de klok gestopt.

```
process KLOK is
let Klok     :klok,
    StartTyd :list,
    StopTyd  :list,
    Tyd      :list,
    Stop     :bool,
```

```
    Interval :int ,
    Stap     :bool,
    Ack      :str
in
 rec-msg(klok,Klok?,StartTyd?,StopTyd?,Interval?,Stap?) .
 Tyd := StartTyd .
 snd-eval(Klok,drukAf(Tyd)) .
  rec-value(Klok,Ack?) .
 snd-msg(klok,ok) .

 Less(StopTyd,Tyd,Stop?).        %% Stop := StopTyd<Tyd
 (if not(Stop) then
   AddSec(Tyd,Interval,Tyd?) .   %% Tyd  := Tyd + Interval
   if Stap then
     snd-eval(Klok,stap) .
      rec-value(Klok,Ack?)
   else
     tau delay(Interval)
   fi .
   snd-eval(Klok,drukAf(Tyd))
   rec-value(Klok,Ack?) .
   snd-note(tyd(Tyd)) .
   Less(StopTyd,Tyd,Stop?)      %% Stop := StopTyd<Tyd
  fi
 )*
 if Stop then snd-terminate(Klok, stop(StopTyd,Tyd)) fi
endlet
```

Van de volgende hulp-processen geven we slechts de headers:

```
process Less       (Tyd0:list, Tyd1:list, Bool:bool?) is ...
process LessMinuut (Tyd0:list, Tyd1:list, Bool:bool?) is ...
process AddSec     (Tyd0:list, Sec:int,   Tyd1:list?) is ...
```

Uiteindelijk geven we de declaratie van de scripts, en een mogelijke initialisatie van de het CDB:

```
tool klok     is {command = "wish-adapter -script tcl.klok      "}
tool ui       is {command = "wish-adapter -script tcl.ui        "}
tool detect   is {command = "wish-adapter -script tcl.detect    "}

tool toewz    is {command = "perl-adapter -script perl.toewz    "}
tool maakcons is {command = "perl-adapter -script perl.maakcons "}
tool instruct is {command = "perl-adapter -script perl.instruct "}

toolbus(CDB([7,00,0],[24,0,0],30,15,60),KLOK,CONTROLLER,INSTRUCT,MAAKCONS,UI)
```

5 Een klein voorbeeld van lineair programmeren

In deze sectie schetsen we in het kort hoe de lineaire ongelijkheden voor de optimalisatie worden verkregen. Voor tekstboeken over lineair programmeren verwijzen wij naar [Wil90] en [Sha90].

De introductie van *toewijzings-variabelen*

Bij het opstellen van de ongelijkheden spelen de reeds toegewezen perrons uit de huidige perron-toewijzing geen rol (afgezien van de bepaling voor de referentie-perrons). We gaan uit van een drietal ritten, met de volgende informatie:

Rit	Refer.Perr	Starttijd Res.	Lengte Res.	Halt.Tijd	Waarde Status
1	2	10.00	4	2	2
2	2	10.01	10	8	1
3	1	10.05	6	5	1

Voor elke rit wordt bepaald aan welke perrons de rit toegewezen kan worden. Laten we aannemen dat er drie perrons zijn, waar elke rit op geplaatst kan worden. Voor elke mogelijke combinatie van rit r en perron p wordt een boolean variabele tr_p geïntroduceerd, dit geeft de volgende variabelen:

t1_1	t1_2	t1_3
t2_1	t2_2	t2_3
t3_1	t3_2	t3_3

Elk van deze variabele zal in het vervolg een waarde krijgen, hetzij 1 of 0. Als de variabele t1_2 de waarde 1 krijgt, dan betekent dat rit 1 aan perron 2 is toegewezen. In het vervolg heten deze variabelen *toewijzings-variabelen*.

De kostenfunctie

In Sectie 2.4 hebben is de kostenfunctie al behandeld die gebaseerd is op de afstand van een toegewezen perron tot het referentieperron, vermenigvuldigd met de status. Dat levert de volgende expressie voor de kostenfunctie:

$$\begin{array}{llllll} & 2\ \text{t1\_1} & + & 0\ \text{t1\_2} & + & 2\ \text{t1\_3} \\ + & 1\ \text{t2\_1} & + & 0\ \text{t2\_2} & + & 1\ \text{t2\_3} \\ + & 0\ \text{t3\_1} & + & 1\ \text{t3\_2} & + & 2\ \text{t3\_3} \end{array}$$

Immers, het referentie-perron van rit 1 is perron 2, dus als rit 1 aan perron 2 toegewezen wordt (en t1_2 dus de waarde 1 heeft, en t1_1 en t1_3 beide 0), dan levert dat geen kosten op. Echter, zodra rit 1 op perron 1 geplaatst wordt, dan levert dat 2 (waarde status × afstand tot referentie-perron) aan kosten op.

Merk op dat, in navolging van de conventies van Cplex, er tussen de toewijzingsvariabelen en hun vermenigvuldigingsfactor geen ×-teken geplaatst wordt.

In het algemeen heeft de kostenfunctie een volgende vorm:

$$\sum_{r:\text{Rit}} \sum_{p:\text{Perron met} toegestaan(r,p)} waarde(status(r)) \times afstand(p, ref\_per) \quad \text{t}r\_p$$

Het lineair programmerings-algoritme zal díe toewijzing opleveren, die het meest optimaal is t.o.v. deze kostenfunctie.

De condities m.b.t. de lengtes

Vervolgens moeten de condities worden opgesteld, die de randvoorwaarden uitdrukken voor de toewijzing.

Een eerste conditie is dat de gezamelijke lengtes van de ritten die op hetzelfde moment op een perron kunnen staan, niet de lengte van het perron mag overschrijden. We gaan uit van de volgende lengtes:

lengte dienstwagens	lengte perrons
rit 1 : 12 meter	perron 1 : 30 meter
rit 2 : 24 meter	perron 2 : 40 meter
rit 3 : 24 meter	perron 3 : 40 meter

Dit houdt in dat de ritten 1 en 2 niet tegelijkertijd op perron 1 kunnen staan, wat uitgedrukt wordt door de volgende conditie:

12 t1_1 + 24 t2_1 <= 30

De lengte van rit 1 (12 meter) wordt alleen meegenomen als t1_1=1, d.w.z. als rit 1 aan perron 1 is toegewezen. Hetzelfde geldt voor rit 2. Op zich zou hier de conditie t1_1 + t2_1 <= 1 ook voldoende geweest zijn, maar in het algemeen, als er meerdere ritten in het spel zijn, is de bovenstaande conditie algemener en daardoor gemakkelijker te verkrijgen.

De overeenkomende conditie voor de ritten 1 en 2, en het perron 2 luidt alsvolgt:

12 t1_2 + 24 t2_2 <= 40

Deze conditie legt echter geen beperkingen op. Deze "neutrale" condities zullen daarom niet gegenereerd worden.

Op gelijke wijze verkrijgen we dat de ritten 2 en 3 niet gezamelijk op de perrons 1,2 en 3 mogen staan:

24 t2_1 + 24 t3_1 <= 30
24 t2_2 + 24 t3_2 <= 40
24 t2_3 + 24 t3_3 <= 40

Merk op dat het zeker is dat rit 1 al vertrokken is als rit 3 arriveert. Daarom genereren de ritten 1 en 3 samen geen condities.

De condities m.b.t. paren van ritten

Vervolgens moeten er voor alle paren van ritten die tegelijkertijd op een perron aanwezig kunnen zijn afgedwongen worden dat:

- de aankomst- resp. vertrekintervallen niet overlappen,
- als de ene rit eerder aankomt, dan moet deze rit ook eerder vertrekken.

In ons voorbeeld overlappen de aankomstintervallen van de ritten 1 en 2. Voor deze ritten moet dus voor elk perron afgedwongen worden dat er maar hoogstens één van beide ritten op geplaatst wordt:

t1_1 + t2_1 <= 1
t1_2 + t2_2 <= 1
t1_3 + t2_3 <= 1

De concrete invoer voor Cplex en de uitvoer

Het invoer voor Cplex van het bovenstaande probleem luidt alsvolgt:

```
Minimize
+ 2 t1_1 + 0 t1_2 + 2 t1_3
+ 1 t2_1 + 0 t2_2 + 1 t2_3
+ 0 t3_1 + 1 t3_2 + 2 t3_3

Subject To
+ t1_1 + t1_2 + t1_3 = 1
+ t2_1 + t2_2 + t2_3 = 1
+ t3_1 + t3_2 + t3_3 = 1
12 t1_1 + 24 t2_1 <= 30
24 t2_1 + 24 t3_1 <= 30
24 t2_2 + 24 t3_2 <= 40
24 t2_3 + 24 t3_3 <= 40
Integer
t1_1
t1_2
t1_3
t2_1
t2_2
t2_3
t3_1
t3_2
t3_2
```

De conditie `+ t1_1 + t1_2 + t1_3 = 1` geeft aan dat rit 1 op precies één perron moet komen te staan. De toewijzingsvariabelen worden gedeclareerd door ze op te geven na het keyword `integer`.

De uitvoer van Cplex is:

```
t1_2                          1.000000
t2_2                          1.000000
t3_1                          1.000000
All other variables in the range 1-9 are zero.
```

6 Conclusies en opmerkingen

Dit stuk beschrijft een model van een Compact Dynamisch Busstation, gebaseerd op een toewijzing die gedurende de looptijd constant aangepast wordt, zodat er bij aankomst van een rit al een geschikt perron bekend is. Als verschillende reserveringen van ritten met elkaar in conflict zijn, dan wordt er m.b.v. lineaire programmeringstechnieken een nieuwe toewijzing opgesteld.

Het model is uitgewerkt in een prototype, dat real-time de detecties van de ritten opvraagt en verwerkt, en zonodig de toewijzing aanpast. Het is uitgewerkt als Toolbus script ([BK95]), en een aantal losse tools. Dit maakt het mogelijk om op eenvoudige wijze verschillende scenarios te executeren. De performance lijkt in de beoogde orde.

Tot slot geven we nog een aantal vragen opmerkingen.

Het verband tussen een huidige status en een detectie

Het is niet duidelijk of er foutmeldingen gegeven moeten worden indien een rit gedetecteerd wordt met een andere status dan de verwachte, bijv. wat te doen als er een rit bij toevoerweg 2 vooraangekondigd wordt, terwijl hij al een status *vooraangekondigd-weg-1*, of zelfs de status *op-perron*, heeft. Een bijzonder eenvoudige oplossing is om dergelijke detecties te negeren.

Een zorgvuldige analyse is echter wel gewenst. Het moet immers wel mogelijk zijn dat een rit, die nog niet vooraangekondigd is, bij de ingang van het station gedetecteerd wordt; in dit geval moet de detectie wel degelijk verwerkt worden, ook al is het gebruikelijk dat zo'n rit eerst vooraangekondigd wordt.

Het vertragen van ritten in de buffer

In Sectie 2.8 is besproken dat een rit die in de buffer wacht totdat een andere rit vertrekt van een perron, als vertraagd beschouwd wordt wanneer die andere rit niet vertrekt van dat perron. Een andere oplossing zou echter zijn dat er op dat moment zo mogelijk een ander perron gekozen wordt, en de rit niet vertraagd wordt.

De omloop en aparte uitstaphaltes

De *omloop* op een station geeft de bewegingen aan van de bussen over het station. Een uitstaphalte is een halte die de bus passeert bij aankomst op het station, om de passagiers te laten uitstappen. Meestal zijn de daartoe bestemde uitstapperrons groot genoeg, en is het niet noodzakelijk om hiervoor reserveringen bij te houden.

In dit stuk zijn de omloop en de uitstaphaltes niet behandeld. Waarschijnlijk is niet bijzonder moeilijk om dit verder in het model uit te werken. Een mogelijkheid is om bij te houden of een rit een *opvolger* heeft, bijv. als een dienstwagen twee ritten achter elkaar rijdt. Nadat een rit gedetecteerd is, of vertraagd, kan dan ook worden nagegaan of dit consequenties heeft voor zijn eventueel opvolgende rit.

Het inconsistent blijven van de toewijzing

Het is mogelijk binnen de daartoe beschikbare tijd er geen nieuwe optimale toewijzing opgesteld kan worden. Het huidige model kent daar nog geen bevredigende oplossing voor.

Het is denkbaar dat in zo'n geval de dienstleider de mogelijkheid moet krijgen om de reserveringen van de ritten in tijd te verschuiven, dan wel op van een aantal ritten de status te veranderen (bijv. een rit laten uitvallen).

Dit betekent een uitbreiding aan de user-interface. Voor het Toolbus-script zelf betekent het dat het proces MaakCons een `niet-geslaagd` boodschap verstuurt aan het Controller proces. Vervolgens geeft de Controller dit door aan het nog te ontwerpen interface-tool, waarna eventuele wijzigingen ingelezen kunnen worden. Uiteindelijk wordt dan MaakCons weer gevraagd om deze nieuwe toewijzing weer consistent te maken.

Eventuele beperkingen van lineaire programmeringstechnieken

In dit stuk is een aantal consistentie-eisen en een aantal elementen van een kostenfunctie besproken. Het is niet duidelijk of elke mogelijke uitbreiding, die noodzakelijk is om een reëel CDB vorm te geven, geformuleerd kan worden in een kostenfunctie en een aantal lineaire ongelijkheden.

Een voorbeeld van zo'n mogelijke uitbreiding is om het doorschuiven van ritten te beperken. Immers, vlak voor vertrek als de passagiers aan het instappen zijn, is het niet wenselijk als een bus moet oprijden, omdat de ervoor geplaatste bus vertrekt.

Deze eis is niet zonder meer in lineaire ongelijkheden uit te drukken, en moet verder worden onderzocht.

Overigens is deze eis conceptueel ook niet zo duidelijk, daar het vertrektijdstip op voorhand niet vastligt, maar ergens in het *vertrekinterval* ligt, zie Sectie 2.4. Als een rit tussen 10.05 en 10.07 vertrekt, en de eis is dat er twee minuten voor vertrek niet meer doorgeschoven wordt, dan betekent dat dus dat de rit vanaf 10.03 niet meer kan doorschuiven. Stel nu dat de rit aankomt tussen 10.00 en 10.02, bijv. precies om 10.02, dan kan de rit maar één minuut doorschuiven.

Kortom, deze eis maakt het CDB veel minder dynamisch, en dus ook minder compact. Ook deze aspecten moeten verder worden onderzocht.

Een meer efficiënt maar minder optimaal algoritme

Het centrale algoritme is het opstellen van de kostenfunctie en de condities, en vervolgens de lineaire optimalisatie door Cplex. Daar de rekentijd niet willekeurig lang kan zijn, is deze begrensd (tot bijv. 5 sec.), waardoor niet optimale oplossingen afgeleverd kunnen worden, zie Sectie C.6.

Waarschijnlijk kan een algoritme opgesteld worden dat, op basis van de ritten met gewijzigde reserveringen, in korte tijd de noodzakelijke wijzigingen (ritten naar ander perron) berekend. Zo zijn er in de meeste gevallen geen wijzigingen noodzakelijk, terwijl in het huidige algoritme wel het hele (relatief tijdrovende) algoritme wordt afgelopen.

Om te bepalen of zo'n algoritme de juiste wijzigingen oplevert, kan het vergeleken worden met het huidige algoritme zonder tijdsbegrenzing (altijd de optimale oplossing), en met tijdsbegrenzing.

References

[BK95] J.A. Bergstra and P. Klint. The discrete time Toolbus. Report P9502, Univers. van Amsterdam, Amsterdam, 1995.

[CPL] CPLEX Optimization Inc. Manual of CPLEX.

[DOWA94] Afdeling Verkeer Dienst Openbare Werken Apeldoorn, 1994.

[Ous94] J.K. Ousterhout. *Tcl/Tk.* Addison Wesley, 1994.

[Sha90] R.D. Shapiro. *Optimization Models for Planning and Allocation.* Wiley & Sons, New York, 1990.

[Wil90] H.P. Williams. *Model Building in Mathematical Programming.* Wiley & Sons, New York, 1990.

[WS90] L. Wall and R.L. Schwartz. *Programming PERL.* UNIX programming. O'Reilly & Associates, Inc, 1990.

In deze appendix zullen de codes, geschreven in de taal Perl, van de tools `perl.toewz`, Sectie B, en `perl.MaakCons`, Sectie C, worden beschreven. De overige tools definiëren de verschillende interfaces en zijn sterk afhankelijk van de taal TCL/TK, en worden daarom hier niet behandeld. Bepaalde gegevens zijn geïnspireerd op het ontwerp van het station Apeldoorn, [DOWA94]. Ook wordt de bibliotheek met basisfuncties gegeven.

A Een aantal algemene opmerkingen over de Perl-codes

A.1 Notatie-conventies

In Perl beginnen alle variabelen met een dollarteken ($). Een voorbeeld van een functiedefinitie is als volgt:

```
sub suc {
local($n)=@_;
 n+1
}
```

Met `local(...)` kunnen één of meerdere variabelen lokaal gedefinieerd worden. Het symbool `@_` bevat al de invoer-argumenten; de regel `local($n)=@_` komt overeen met de declaratie van de formele argumenten in de functie-header zoals gebruikelijk in andere programmeertalen.

De functie kan worden aangeroepen door het te laten voorafgaan door het &-teken, bijv. `$x = &suc(1)`.

Assignments worden aangeduid met een enkel =-teken, de test op gelijkheid daarintegen met een dubbel ==-teken, en de test op ongelijkheid met =!.

Alles na het #-teken is commentaar.

A.2 De declaratie van de datatypen

De taal Perl kent zelf geen (data)typen; het kent slechts termen die afhankelijk van de context als strings dan wel als integers geïnterpreteerd worden.

Als toelichting op de te presenteren Perl-codes worden hier een aantal typen informeel geïntroduceerd. Ook kunnen deze typen een meer expliciete rol spelen als de codes worden omgezet naar een getypeerde taal als bijv. C.

```
RIT             = DIGIT[7]
PERRON          = INT
BUFFER          = INT
TYD             = NUM[2]","NUM[2]","NUM[2]
UUR             = INT
MIN             = INT
SEC             = INT
STATUS          =   {"1","2","vertraagd","vooraangekondigd",
                    "gearriveerd","opPerron","vertrokken"}
                    || "opBuffer"BUFFER
                  # Vereenvoudigde verzameling van status-waarden,
                  # vooraangekondigd i.p.v. vooraangekondigd-T
                  # gearriveerd i.p.v. vooraangekondigd,
```

```
                # idem voor gearriveerd-I en vertrokken-U,
                # zie ook Sectie 2.7
LYNNR           = DIGIT[4]
ROUTE           = DIGIT[4]
WAGEN           = DIGIT[4]
RITTEN          = list-of(RIT)
PERRON-TOEWYZING = list-of(RIT":"PERRON","PERRON","SEC","SEC","SEC","STATUS
                                ","LYNNR","ROUTE","WAGEN)
HALTE           = INT
VERLOOP         = list-of(HALTE","SEC)
HALTE-TOEWYZING = list-of(RIT":"PERRON","PERRON","SEC","SEC","SEC","STATUS
                                ","VERLOOP","LYNNR","ROUTE","WAGEN)

DETECTIE        = RIT":"TYD","TYD","TYD","LYNNR
DETECTIES       = list-of(DETECTIE)
DETECTIE-PUNT   = {"1"} # Voorlopig slechts \'e\'en detectiepunt
```

We gaan ervan uit dat elk type, RIT, SEC, etc. een constante `$nil` kent. Elk array wordt impliciet geïnitialiseerd zodat het op zijn gehele domein `$nil` aflevert. Voor een lijst-soort geldt dat `$nil` overeenkomt met `$leeg`, en dus levert `&isIn($Rit,$nil)` altijd `$false` op.

A.3 De declaratie van globale constanten

```
$false                  = (1==0);
$true                   = (1==1);
$leeg                   = '';

$minPerron              = 1;
$maxPerron              = 11;
$reservePerron          = $maxPerron;
$nilPerron              = $minPerron-1;

$minBuffer              = 1;
$maxBuffer              = 16;

$gers1Status            = "1";
$gers2Status            = "2";
$vrtrgStatus            = "vertraagd";
$gedetStatus            = "vooraangekondigd";
$gearrStatus            = "gearriveerd";
$opPerStatus            = "opPerron";
$vrtrkStatus            = "vertrokken";

$gers1Waarde            = 1;
$gers2Waarde            = 2;
$vrtrgWaarde            = 3;
$gedetWaarde            = 4;
$gearrWaarde            = 5;
```

```
$opBufWaarde            = 6;
$opPerWaarde            = 7;
$vrtrkWaarde            = 0;

$standaardDienstwagen   = "0000";
$stadRoute              = "0000";
$streekRoute            = "1111";
```

B Het Toewijzings-tool

B.1 De declaratie van de globale variabelen

De essentiële informatie waarin de toewijzing wordt bijgehouden

Het tool `perl.toewz` houdt de gehele toewijzing bij in een aantal arrays; waarbij het haltetoewijzing van type `HALTETOEWYZING`, zie Sectie A.2, ontleedt en opslaat in afzonderlijke arrays.

Zo is er een array [3] `$PO`; `$PO{$Rit}` geeft het perron aan waaraan de rit `$Rit` is toegewezen.

```
$PO      {RIT} -> PERRON  # Toegewezen perron v/e rit
$P1      {RIT} -> PERRON  # Voorkeursperron v/e rit
$T       {RIT} -> SEC     # Begintijdstip van reservering v/e rit
$H       {RIT} -> SEC     # Halteertijd v/e rit
$R       {RIT} -> SEC     # Lengte van de reservering v/e rit
$S       {RIT} -> STATUS  # Status v/e rit
$Lyn     {RIT} -> LYNNR   # Lynnummer v/e rit
$Route   {RIT} -> ROUTE   # Route v/e rit
$Wagen   {RIT} -> WAGEN   # Wagen v/e rit
```

Afleidbare informatie

De bovenstaande arrays bevatten al de essentiële informatie. In sommige gevallen is bepaalde afleidbare informatie noodzakelijk, bijv. van welke ritten de reservering op een bepaald moment begint. Daartoe worden de volgende arrays bijgehouden, zodat de informatie snel voorhanden is en niet steeds berekend hoeft te worden uit de bovenstaande arrays:

```
$StartReserv      {SEC}    -> RITTEN # Ritten waarvan de reservering
                                     # start op een bepaald tijdstip.
$AlGereserv       {SEC}    -> RITTEN # Ritten waarvan de reservering
                                     # al is ingegaan op een bepaald tijdstip.
$EindeReserv      {SEC}    -> RITTEN # Ritten waarvan de reservering
                                     # afloopt op een bepaald tijdstip.
$VerwVoorAank     {SEC}    -> RITTEN # Ritten die op een bepaald moment
                                     # verwacht te worden vooraangekondigd
$VerwVoorAankTyd {RIT}     -> SEC    # Verwachte tijdstip van vooraankomst
```

[3] In Perl zelf staat `$PO{$Rit}` voor een variabele; de gehele array wordt aangeduid met `%PO`. In dit document zullen we dat onderscheid echter zo min mogelijk maken; alle opmerkingen over typen en de typering van arrays hebben vanuit de Perl-codes gezien de status van commentaar.

```
$VerwStatAank     {SEC}     -> RITTEN # Ritten die op een bepaald moment
                                      # verwacht worden bij ingang station
$VerwStatAankTyd {RIT}      -> SEC    # Verwachte tijdstip van stationsaankomst
$Queue            {PERRON} -> RITTEN # De ritten die in aantocht zijn
                                      # (al vooraangekondigd) voor een perron
```

De invarianten over de afleidbare informatie

Een invariant is een logische uitspraak die gedurende de alle stappen van een algoritme waar is, of althans waar zou moeten zijn.

De volgende invariant garandeert dat de arrays hun "naam" waarmaken; `$StartReserv{$t}` bevat precies díe ritten waarbij de reservering op tijdstip `t` begint, `$EindeReserv{$t}` bevat precies díe ritten waarbij de reservering op tijdstip afloopt en `$AlGereserv{$t}` bevat díe ritten waarbij de reservering voor `t` is ingegaan, en na `t` afloopt.

`$VerwVoorAankTyd{$Rit}` is het tijdstip waarop de `Rit` naar verwachting vooraangekondigd wordt. `$VerwStatAankTyd{$Rit}` is het tijdstip waarop de `Rit` naar verwachting op het station aankomt.

Alle ritten die naar verwachting op tijdstip `t` vooraangekondigd worden, worden bijgehouden in `$VerwVoorAank{$t}`, en alle ritten die naar verwachting op tijdstip `t` op het station aankomen, worden bijgehouden in `$VerwStatAank{$t}`.

Bij initialisatie op `$nil` voldoen de arrays aan deze conditie, bij elke volgende aanpassing moet nagegaan worden dat deze invariant gehandhaafd blijft.

```
for each $Rit:RIT, $t:SEC such that $Rit!=$nil && $t!=$nil:

   &isIn($Rit,$StartReserv{$t})  <=> $t == $T{$Rit}
&& &isIn($Rit,$EindeReserv{$t})  <=> $t == $T{$Rit}+$R{$Rit}
&& &isIn($Rit,$AlGereserv{$t})   <=> ($T{$Rit}<$t && $t<$T{$Rit}+$R{$Rit})

&& $VerwVoorAankTyd{$Rit} != $nil => $VerwStatAankTyd{$Rit} == $nil
&& &isIn($Rit,$VerwVoorAank{$t}) <=> $VerwVoorAankTyd{$Rit} == $t
&& &isIn($Rit,$VerwStatAank{$t}) <=> $VerwStatAankTyd{$Rit} == $t
```

B.1.1 De functie `init`

Deze functie stelt de initiële toewijzing op, d.w.z. dat het de globale arrays uit de voorgaande sectie initialiseert.

Als invoer heeft het o.m. de looptijd, gedefinieerd door `$Start` en `$End`, en een `$Horizon`. Alle te introduceren ritten die binnen `$Start` en `$Start+$Horizon` vallen krijgen de "hoge" gereserveerde status `$gersStatus2`, alle andere krijgen de lagere status `$gersStatus1`.

De functie `MaakTydInSec` converteert een tijdstip van de vorm `uur.min.sec` in seconden, en is gedefinieerd in de bibliotheek, zie D.

De informatie over de dienstregeling zelf wordt van file ingelezen.

```
sub init {
 local($Start,$End,$Horizon)=@_; # TYD, TYD, MIN
 ($Start,$End)     = (&MaakTydInSec($Start),&MaakTydInSec($End));
 $Horizon          = 60*$Horizon; # formuleer de horizon in seconden
```

```
local($EindeHor) = $Start+$Horizon; # SEC
local($p,$b);                       # PERRON, BUFFER
```

Initialiseer de vrije ruimte per perron:

```
$p = $minPerron;
while ($p <= $maxPerron) { $vryeRuimte{$p} = &perronLengte($p); ++$p; }
```

Initialiseer elke buffer op leeg:

```
$b = $minBuffer;
while ($b <= $maxBuffer) {$OpBuffer{$b} = $leeg;++$b;}
```

Bepaal interval in hele uren waar looptijd in ligt:

```
local($StartUur,$EndUur) = (int($Start/3600),int($End/3600)+1); # UUR, UUR
```

Declareer overige locale hulpvariabelen:

```
local($l,$p);       # LYNNR, PERRON
local($a,$v,$u);    # MIN (aankomst), MIN (vertrek)
local($u)           # UUR
local(%cnt);        # $cnt{LYNNR.UUR} -> INT
local($t,$t0,$t1); # SEC, SEC, SEC
local($c);          # INT
local($Rit);        # RIT
```

`$cnt{$l.$u}` geeft aan hoeveel ritten van lijn `$l` al zijn opgenomen in uur `$u`. Voor elke `$l` en `$u` is `$cnt{$l.$u}` impliciet op 0 geïnitialiseerd.

Open vervolgens de file `f.dienstRegApeldoornStad`.

```
open(DIENSTREG,"f.dienstRegApeldoornStad");
while(<DIENSTREG>)
```

Deze file bestaat uit regels als `4a:2,32,37`, welke aangeeft dat lijn `4a` voor elk uur in de looptijd naar verwachting van `.32` tot `.37` op perron 2 staat.

Haal telkens een regel op uit deze file, en ontleed deze in resp. het lijnnummer (`$l`), het perron (`$p`), de aankomst (`$a`) en het vertrek (`$v`). Ga voor elk uur tussen `$StartUur` en `$EndUur` na of er voor deze regel een rit geïntroduceerd moet worden, d.w.z. of de rit tussen `$Start` en `$End` valt. Zoja, initialiseer dan de globale arrays `$P0`, `$P1`, etc., en pas `$StartReserv`, `$EindeReserv` en `$AlGereserv` aan, waarbij opgemerkt moet worden dat de invariant hierbij gerespecteerd wordt.

```
{if(/^(\w{4}):(\d+),(\d+),(\d+)$/)
 {($l,$p,$a,$v) = ($1,$2,$3,$4);
  $u       = $StartUur;
  while ($u<=$EndUur)
  {++$cnt{$l.$u};
   $c   = $cnt{$l.$u};
   $t0  = &TydInSec($u,$a,0);
   $t1  = &TydInSec($u,$v,0);
   $t   = &vulNulAan($u);
```

```
   $Rit = $l.$t.$c;
   if (  &inInterval($t0,$Start,$End)
      || &inInterval($t1,$Start,$End)) {
    $P0{$Rit}     = $p;
    $P1{$Rit}     = $p;
    $T{$Rit}      = $t0;
    $H{$Rit}      = $t1-$t0;
    $R{$Rit}      = $H{$Rit}+60*&speling($Rit);
    if (  &inInterval($t0,$Start,$EindeHor)
       || &inInterval($t1,$Start,$EindeHor)) {
     $S{$Rit}     = $gers2Status; }
    else {
     $S{$Rit}     = $gers1Status; }
    $Lyn{$Rit}    = $l;
    $Route{$Rit}  = $stadRoute;
    $Wagen{$Rit}  = $standaardDienstwagen;

    $StartReserv{$t0} = &add($Rit,$StartReserv{$t0});
    $t = $t0+1;
    while ($t<$t0+$R{$Rit}) {
     $AlGereserv{$t} = &add($Rit,$AlGereserv{$t}); ++$t
    }
    $EindeReserv{$t1} = &add($Rit,$EindeReserv{$t1});
   }
   ++$u;
  }
 }  else {if (/^(.)*$/) {
    &Print("Wrong format in f.dienstRegApeldoornStad: $1")}}
}
close DIENSTREG;

local($Detecties) = $leeg;              # DETECTIES
local($Dp,$Int) ;                       # DETECTIE-PUNT, SEC
local($VerwVoorAankTyd,$VoorAankTyd);   # SEC, SEC
local($VerwStatAankTyd,$StatAankTyd;    # SEC, SEC
local($VrtrVertrTyd);                   # SEC
foreach $Rit (sort keys(%P0)) {
```

Voor elke `$Rit` in de gehele toewijzing worden nu random de momenten van detectie opgesteld.

`$VoorAankTyd` is de verwachte vooraankomst-tijd, en `$VerwStatAankTyd` is de verwachte stationsaankomst-tijd. Neem de werkelijke vooraankomst-tijd, `$VoorAankTyd`, random in het interval `[$VerwVoorAankTyd - 0.4*$Int, $VerwVoorAankTyd + 0.6*$Int]`.
De factoren `0.4` en `0.6` dwingen af dat het interval voor 0,4 voor de geplande vooraankomsttijd ligt, en voor 0.6 erna; deze factoren zijn tamelijk willekeurig, samen opgeteld moeten ze wel 1.0 zijn.

Neem de werkelijke stationsaankomst-tijd, `$StatAankTyd`, random in het interval `[$VerwStatAankTyd-30, $VerwStatAankTyd+30]`.
Neem de vertraging van het vertrek, `$VrtrgVertrTyd`, random tussen 0 en 15 (seconden).

Vertragingen in buffers worden niet gemodelleerd, als een rit op een buffer geplaatst is en de in de instructie wordt gegegeven dat de rit moet oprijden naar het perron, dan wordt de rit geacht direct op het perron plaats te nemen. Dit kan verder uitgewerkt worden door ook detectiepunten aan het begin van het perron in ogenschouw te nemen.

Pas uiteindelijk de arrays aan die betrekking hebben op de verwachte vooraankomst.

```
  $Dp                = &detectiePunt($Rit);
  $Int               = &aankomstInterval($Rit);
  $VerwVoorAankTyd   = $T{$Rit} - &afstandInTyd($Dp);
  $VoorAankTyd       = $VerwVoorAankTyd - int(0.4*$Int) + int(rand($Int));
  $VerwStatAankTyd   = $VoorAankTyd + &afstandInTyd($Dp);
  $StatAankTyd       = $VerwStatAankTyd - 30 + int(rand(60));
  $VrtrgVertrTyd     = int(rand(15));
  $Dt{$Rit}          = "$VoorAankTyd,$StatAankTyd,$VrtrgVertrTyd";
  $Detecties         = &add("$Rit:$Dt{$Rit},$Lyn{$Rit}",$Detecties);
  $t                 = $T{$Rit}-&afstandInTyd($Dp);
  $VerwVoorAank{$t} = &add($Rit,$VerwVoorAank{$t});
  $VerwVoorAankTyd{$Rit} = $t;
 }
```

Stuur vervolgens de opgebouwde detecties naar de Toolbus.

```
 do TBsend("snd-value(\"$Detecties\")");
}
```

B.2 De functie VerwerkDetecties

De functie krijgt alle detecties binnen, en verwerkt deze; de verschillende onderdelen worden hieronder in subsecties behandeld.

B.2.1 De header

```
sub verwerkDetecties {
local($Nu,$VoorAank,$StatAank,$PerVertr)=@_; # SEC, RITTEN, RITTEN, RITTEN

local($Rit,$Ritten); #RIT RITTEN
local($p,$b,$s);     # PERRON, BUFFER, SEC
```

Het huidig tijdstip is `$Nu`, `$VoorAank` bevat de ritten die vooraangekondigd zijn, `$StatAank` bevat de ritten die bij het station zijn aangekomen, en `$PerVertr` bevat de ritten die van hun perron vertrokken zijn.

B.2.2 De vooraangekomen ritten

Een vooraangekomen rit wordt eerst verwijderd uit de arrays die betrekking hebben op de te verwachten vooraankomsten.

Dan wordt elke vooraangekomen rit toegevoegd aan de `$Queue` van zijn toegewezen perron en zonodig wordt de status aangepast. Deze `$Queue` per perron wordt bijgehouden om later de instructies te kunnen opstellen.

Vervolgens wordt bijgehouden dat de rit ieder moment bij het station kan arriveren (`$VerwStatAankTyd{$Rit}=$Nu`).

Als de rit echter al is vooraangekondigd, en `$VerwVoorAankTyd{$Rit}` ongedefinieerd is, dan wordt deze rit overgeslagen, en een foutmelding gegenereerd.

```
#Verwerk de vooraankomsten:
while ($VoorAank ne $leeg)
 {($Rit,$VoorAank)       = &remTop($VoorAank);
  if ($VerwVoorAankTyd{$Rit}) {   # if ($VerwVoorAankTyd{$Rit} != $nil)
   $s                     = $VerwVoorAankTyd{$Rit};
   $VerwVoorAank{$s}      = &rem($Rit,$VerwVoorAank{$s});
   undef                    $VerwVoorAankTyd{$Rit}; # zet op $nil
   $p                     = $P0{$Rit};
   $Queue{$p}             = &add($Rit,$Queue{$p});
   if (&waarde($S{$Rit})<&waarde($gedetStatus))
     {$S{$Rit}             = $gedetStatus;}
   $VerwStatAank{$Nu}     = &add($Rit,$VerwStatAank{$Nu});
   $VerwStatAankTyd{$Rit} = $Nu;
  } else {
   &Print("Rit $Rit is al vooraangekondigd!\n");}
 }
```

B.2.3 De op het station aangekomen ritten

Als een op het station aangekomen rit nog niet vooraangekondigd was (en `$VerwVoorAankTyd{$Rit}` dus gedefinieerd is), dan wordt de rit eerst verwijderd uit de arrays die betrekking hebben op de verwachte vooraankomsten, en wordt de rit alsnog in de `$Queue` van zijn toegewezen perron geplaatst.

Vervolgens wordt de status aangepast, en wordt de rit verwijderd uit de arrays die betrekking hebben op de station-aankomsten.

```
while ($StatAank ne $leeg)
 {($Rit,$StatAank)       = &remTop($StatAank);
  if ($VerwVoorAankTyd{$Rit}) {
   $s                = $VerwVoorAankTyd{$Rit};
   $VerwVoorAank{$s} = &rem($Rit,$VerwVoorAank{$s});
   undef               $VerwVoorAankTyd{$Rit};
   $p                = $P0{$Rit};
   $Queue{$p}        = &add($Rit,$Queue{$p});
  }
  $S{$Rit}               = $gearrStatus;
  $s                     = $VerwStatAankTyd{$Rit};
  $VerwStatAank{$s}      = &rem($Rit,$VerwStatAank{$s});
  undef $VerwStatAankTyd{$Rit};
 }
```

B.2.4 De vertrekkende ritten

Van elke vertrekkende rit wordt de status aangepast, evenals de `$vryeRuimte` van zijn perron.

```
while ($PerVertr ne $leeg)
 {($Rit,$PerVertr)       = &remTop($PerVertr);
  if ($S{$Rit} == $opPerStatus) {
    $p                   = $PO{$Rit};
    $vryeRuimte{$p}      = $vryeRuimte{$p} + &lengte($Rit);
  } else {&Print("Rit $Rit is vertrokken, maar stond niet op perron!!\n");}
  $S{$Rit}               = $vrtrkStatus;
 }
```

B.2.5 Het geven van instructies n.a.v. detecties

Nadat de detecties zelf verwerkt zijn, moeten er instructies gegenereerd worden. De lijst `$NaarPerron` bevat paren `RIT":"PERRON`, van ritten die op moeten rijden naar een bepaald perron. De lijst `$NaarBuffer` bevat paren `RIT":"BUFFER`, van ritten die op moeten rijden naar een bepaalde buffer.

Nadat een rit naar een perron is opgereden kan pas bepaald worden wanneer hij naar verwachting zal vertrekken. Deze informatie wordt bijgehouden in het array `$VerwVertr` (verwachte vertrek) en de lijst `$VerwVanPerron`. Deze laatste lijst zal worden opgestuurd naar de Toolbus.

Eerst worden voor elk perron alle in aantocht zijnde ritten (bijgehouden in `$Queue`) geordend op aankomsttijd en vertrektijd. Vervolgens worden deze ritten in volgorde afgelopen.

Als een rit net is gearriveerd, of al in een buffer staat, en er is ruimte vrij op het toegewezen perron, dan kan hij daarnaar oprijden. De `$vryeRuimte` van het perron wordt aangepast, evenals de bovengenoemde lijsten en het bovengenoemde array.

Als de rit wel in de `$Queue` zit, maar hij moet niet oprijden naar een perron, dan wordt hij weer in de `$Queue` teruggeplaatst. Als de rit al wel is gearriveerd dan wordt er een vrije buffer gezocht.

```
local($NaarPerron) = $leeg; # list-of(RIT":"PERRON)
local($NaarBuffer) = $leeg; # list-of(RIT":"BUFFER)
local($VerwVanPer) = $leeg; # list-of(RIT":"PERRON","SEC)

local($p) = $minPerron;
while ($p <= $maxPerron) {
 $Queue{$p}              = &orden($Queue{$p});
 $Ritten                 = $leeg;

 while ($Queue{$p} ne $leeg) {
  ($Rit,$Queue{$p}) = &remTop($Queue{$p});
  if ( ( $S{$Rit} eq $gearrStatus
       || &waarde($S{$Rit}) == $opBufWaarde )
     && ($vryeRuimte{$p} >= &lengte($Rit) )  ) {
   $vryeRuimte{$p}       = $vryeRuimte{$p} - &lengte($Rit);
   $NaarPerron           = &add("$Rit:$p",$NaarPerron);
   # verwachte vertrektijd :
   $t                    = &max($Nu,$T{$Rit})+$H{$Rit};
   $VerwVanPer           = &add("$Rit:$p,$t",$VerwVanPer);
   $VerwVertr{$t}        = &add($Rit,$VerwVertr{$t});
   $S{$Rit}              = $opPerStatus;
  } else {
```

```
    $Ritten              = "$Ritten$Rit&";
    if ($S{$Rit} eq $gearrStatus ) {
     $b = $minBuffer;
     while ($b <= $maxBuffer && $OpBuffer{$b} ne $leeg) {++$b;}
     if ($OpBuffer{$b} eq $leeg) {
       $OpBuffer{$b}     = $Rit;
       $NaarBuffer       = &add("$Rit:$b",$NaarBuffer);
       $S{$Rit}          = "opBuffer$b";
       } else {&Print("Alle buffers zijn vol!!!
                      $Rit niet op buffer geplaatst.");}
     }
    }
   }
  $Queue{$p} = $Ritten;++$p;
  }
do TBsend("snd-value(opRydInstr(\"$NaarPerron\",
                                \"$NaarBuffer\",\"$VerwVanPer\"))");
}
```

B.3 De functie stelVertrekInstrOp

Deze functie heeft twee tijdstippen als argument, $Vorig en $Nu. Het stelt in $VertrVanPerron een lijst van paren $Rit,$p op. Elk paar geeft aan dat de betreffende rit van het perron moeten vertrekken.

Als een rit tussen $Vorig en $Nu zou moeten vertrekken, maar niet (meer) aanwezig is, dan wordt er een foutmelding gegeven.

```
sub stelVertrekInstrOp {
local($Vorig,$Nu)=(@_);
($Vorig,$Nu) = (&MaakTydInSec($Vorig),&MaakTydInSec($Nu));

 local($t,$VertrVanPerron,$Ritten,$Rit);

 $VertrVanPerron = $leeg;
 while ($t<=$Nu) {
  $Ritten = $VerwVertr{$t};
  while ($Ritten ne $leeg) {
   ($Rit,$Ritten) = &remTop($Ritten);
    if ($S{$Rit} eq $opPerStatus) {
     $VertrVanPerron = &add("$Rit:$PO{$Rit}",$VertrVanPerron);
    } else {&Print("$Rit moet vertrekken van $PO{$Rit},
                    maar is niet aanwezig; S{$Rit}:$S{$Rit}"); }
  }++$t;
 }
do TBsend("snd-value(\"$VertrVanPerron\")");
}
```

B.4 De functie bepaalVertragingen

De functie levert de lijst $TotaalVertraagd van vertraagde ritten af, d.w.z. alle ritten die tussen $Vorig en $Nu naar verwachting vooraangekondigd hadden zullen worden, dan wel naar verwachting bij het station zouden aankomen.

```
sub bepaalVertragingen {
local($Vorig,$Nu)=(@_);           # SEC, SEC
local($Rit,$Vertraagd,$Interval); # RIT, RITTEN, SEC
local($TotaalVertraagd)=$leeg;    # RITTEN

 local($t,$N) = ($Vorig+1,$Nu+1); # SEC, SEC

 while ($t<=$Nu) {
```

Per tijdstip $t wordt eerst bepaald welke ritten er op dat moment hadden zullen aankomen ($VerwVoorAank{$t}). Voor elke rit in deze lijst wordt de status zo nodig aangepast. Als de rit al vast staat, dan volgt er een foutmelding. Zoniet, dan wordt de reservering verlengd, en wordt de rit aan de uitvoer toegevoegd. Ook wordt de verwachte vooraankomsttijd verschoven naar $N, het eerstvolgende moment.

```
  $Vertraagd         = $VerwVoorAank{$t};
  $VerwVoorAank{$t}  = $leeg;
  while ($Vertraagd ne $leeg) {
   ($Rit,$Vertraagd)  = &remTop($Vertraagd);
   if ($S{$Rit} eq $gers1Status || $S{$Rit} eq $gers2Status)
    { $S{$Rit} = $vrtrgStatus;}
   if (&vast($S{$Rit})) {
     &Print("Rit $Rit staat vast, status $S{$Rit}!!
                     Geen vertraagde vooraankomst.")
   } else {
   $Interval         = $Nu - $t;
   &VerlengReserv($Nu,$Rit,$Interval);
   $TotaalVertraagd  = &add($Rit,$TotaalVertraagd);
   $VerwVoorAank{$N} = &add($Rit,$VerwVoorAank{$N});
   $VerwVoorAankTyd{$Rit} = $N;
   }
  }
```

De ritten met een vertraagde stationsaankomst worden idem behandeld.

```
  $Vertraagd         = $VerwStatAank{$t};
  $VerwStatAank{$t}  = $leeg;
  while ($Vertraagd ne $leeg) {
   ($Rit,$Vertraagd)  = &remTop($Vertraagd);
   if (&vast($S{$Rit})) {
    &Print("Rit $Rit staat vast, status $S{$Rit}!!
                     Geen vertraagde stationaankomst.")
```

```
   } else {
    $Interval       = $Nu - $t;
    &VerlengReserv($Nu,$Rit,$Interval);
    $TotaalVertraagd =  "$TotaalVertraagd$Rit&";
    $VerwStatAank{$N} = "$VerwStatAank{$N}$Rit&";
    $VerwStatAankTyd{$Rit} = $N;
   }
  }

 ++$t;
 }
```

Uiteindelijk wordt de opgestelde lijst `$TotaalVertraagd` naar de Toolbus gestuurd.

```
do TBsend("snd-value(\"$TotaalVertraagd\")");
}
```

B.5 De functie `schuifHorizonOp`

Deze functie past de status aan van alle ritten met een "lage" gereserveerde status 1 (`$gers1Status`) die inmiddels binnen de horizon `$Hor` zijn komen te vallen; de status wordt "verhoogd" tot 2 (`$gers2Status`).

```
sub schuifHorizonOp{
local($OudStart,$NwStart,$Hor)=@_;
$OudStart = &MaakTydInSec($OudStart);
$NwStart  = &MaakTydInSec($NwStart);

local($t,$Rit,$Verwacht);
local($Nieuw)=$leeg; #Hierin worden alle nieuwe ritten bijgehouden.

 $t = $OudStart;
 while ($t <= $NwStart) {
  $nieuw = $StartReserv{$t+$Hor};
  while ($nieuw ne $leeg) {
   ($Rit,$nieuw) = remTop($nieuw);
    if ($S{$Rit} eq $gers1Status) {
     $Nieuw   = &add($Rit,$Nieuw);
     $S{$Rit} = $gers2Status;
     # Print("Rit:$Rit van status $gers1Status naar $gers2Status.");
    }
  }
  ++$t;
 }
do TBsend("snd-value(\"$Nieuw\")");
}
```

B.6 De function verwerkWijzigingen

Deze functie verplaatst ritten naar een ander perron. Als één van de ritten al vast staat of een van de nieuwe perrons is niet valide, dan wordt geen van de wijzigingen doorgevoerd.

De invoer-lijst $Wyzigingen wordt twee keer doorlopen, doorvoor wordt er eerst een kopie gemaakt in $WyzigingenKopie.

```
sub verwerkWyzigingen {
local($Wyzigingen)=@_;                  # list-of(RIT":"PERRON)
local($Wyz);                            # RIT":"PERRON
local($WyzigingenKopie) = $Wyzigingen; # list-of(RIT":"PERRON)
local($Toegestaan       = $true;        # BOOL

 while ($Toegestaan && $Wyzigingen ne $leeg) {
  ($Wyz,$Wyzigingen) = &remTop($Wyzigingen);
  if ($Wyz =~ /^(\w*):(\d*)$/) {
   ($Rit,$NwPer) = ($1,$2);
   $Toegestaan = (!&vast($Rit)) && &valide($NwPer);
  } else {&Print("Verkeerd formaat wyziging:$Wyz");}
 }

 if ($Toegestaan) {
  $Wyzigingen = $WyzigingenKopie;
  while ($Wyzigingen ne $leeg) {
   ($Wyz,$Wyzigingen) = &remTop($Wyzigingen);
   if ($Wyz =~ /^(\w*):(\d*)$/) {
    $PO{$1} = $2;
   } else {&Print("Verkeerd formaat wyziging:$Wyz");}
  }
 } else {&Print("Wyzigingen kunnen niet doorgevoerd worden;
                 rit $Rit is vast of $NwPer is niet valide!!\n");}

do TBsend("snd-value(\" \")");
}
```

C Het MaakCons-tool

C.1 De declaratie van enkele typen en globale variabelen

In Sectie 5 is uitgelegd dat voor elke mogelijke toewijzing van een rit aan een perron een boolean variabele, ofwel een *toewijzingsvariabele* moet worden geïntroduceerd. Daartoe definiëren we het type TOEWYZ-VAR.

De invoer voor Cplex, het lineair programmerings-algoritme, wordt weggeschreven naar een file, waaruit Cplex zelf weer leest. We introduceren een type STRINGS van lijsten van strings, elke string zal gescheiden door een "newline" worden geprint in de genoemde file.

```
TOEWYZ-VAR      = "t"RIT"_"PERRON
STRINGS         = list-of(STRING)
```

Als rit $Rit op perron $p geplaatst kan worden, dan wordt er een toewijzings-variabele t$Rit_$p geïntroduceerd, wat wordt bijgehouden in het volgende array:

```
$Tws {TOEWYZ-VAR} -> BOOL
```

De input van Cplex bestaat uit een expressie ($Kosten) die de kostenfunctie weergeeft, en uit een grote conditie. Deze conditie bestaat uit drie subcondities: $CondsEenPerron dwingt af dat elke rit op precies één perron geplaatst wordt, $CondsLengte dwingt af dat de gezamelijke lengte van ritten op een perron niet de lengte van het perron overschrijden, en $CondsNietTegelykerTyd sluit alle combinaties van ritten uit die niet gezamelijk op een perron geplaatst mogen worden.

Al deze variabelen zijn van type STRINGS:

```
$Kosten                -> STRINGS
$CondsEenPerron        -> STRINGS
$CondsLengte           -> STRINGS
$CondsNietTegelykerTyd -> STRINGS
```

Ook dit tool kent de datatypen en constanten uit de Secties A.2 en A.3. Lokaal worden dezelfde arrays $P0, $P1, etc. definieerd als door het tool perl.toewz, zoals in Sectie B.1.

Ook kent het de arrays $StartReserv en $EindeReserv, zoals gedeclareerd in Sectie B.1.

C.2 De hoofd-functie maakcons

Deze functie bestaat uit het aanroepen van een aantal functies die het eigenlijke werk doen; eerst wordt de invoer ingelezen, vervolgens worden de condities opgesteld die afdwingen dat de lengtes van de perrons niet wordt overschreven, vervolgens Cplex aangeroepen. Uiteindelijk worden de opgebouwde datastructuren weggegooid, en worden de voorgestelde wijzigingen naar de Toolbus gestuurd.

```
sub maakcons {
local($Toewz,$Vertraagd,$Nieuw) = @_;         # PERRON-TOEWYZING, RITTEN, RITTEN
local($Gewyzigd) = &conc($Vertraagd,$Nieuw); # RITTEN
  if ($Toewz ne $leeg) {
    &LeesInvoer($Toewz);
    &CheckLengte;
    &CheckTegelykertyd;
    &RoepCplxAan($Gewyzigd);
    &CollectGarbage;
   } else {($Wyzigingen,$Geslaagd)=($leeg,$true)}
   if ($Geslaagd=$true) {
    do TBsend("snd-value(\"$Wyzigingen\")");
   } else {
    do TBsend("snd-value(niet-geslaagd)");
   }
}
```

C.3 De functie LeesInvoer

Deze functie leest de opgestuurde perrontoewijzing uit, en initialiseert de globale arrays.

Voor elke rit roept het de functie &stelMogToewzOp aan, die voor elk perron waarop de rit geplaatst kan worden een toewijzings-variabele introduceert.

```
sub LeesInvoer
{local($Toewz)=@_; # PERRON-TOEWYZING
 local($Tw);       # RIT":"PERRON","PERRON","SEC","SEC","SEC","STATUS
                   #         ","LYNNR","ROUTE","WAGEN"
 local($Rit);      # RIT
 local($t0,$t1);   # SEC,SEC

 $CondsEenPerron = $leeg;
 $Kosten         = $leeg;

 while ($Toewz != $leeg)
  {($Tw,$Toewz) = &remTop($Toewz);
   if ($Tw =~ /^(\w*):(\d*),(\d*),(\d*),(\d*),(\d*),(\w*),(\w*),(\w*),(\w*)$/)
   {$Rit = $1;
    ($P0{$Rit},$P1{$Rit},$T{$Rit},$H{$Rit},$R{$Rit},$S{$Rit},
        $Lyn{$Rit},$Route{$Rit},$Wagen{$Rit})
      = ($2,$3,$4,$5,$6,$7,$8,$9,$10);

    local($t0,$t1)=($T{$Rit},$T{$Rit}+$R{$Rit});
    $StartReserv{$t0} = &add($Rit,$StartReserv{$t0});
    $EindeReserv{$t1} = &add($Rit,$EindeReserv{$t1});
   }
  }

 foreach $Rit (keys(%P0)) { &stelMogToewzOp($Rit); }
}
```

De functie &stelMogToewzOp bepaalt eerst op welke perrons de rit mag geplaatst mag worden ($Perrons). Vervolgens introduceert het voor elk van deze perrons een toewijzingsvariabele. De introductie van zo'n toewijzingsvariabele wordt bijgehouden in de globale array $Tws, en in de globale variabelen $Kosten en $CondsEenPerron.

```
sub stelMogToewzOp
{local($Rit)=@_;  # RIT
 local($Kost,$Cond); # STRING, STRING
 local($Perrons,$p); # PERRONS, PERRON
 local($Tw);         # TOEWYZ-VAR
 local($n,$s);       # INT, WAARDE
 local($RefPerron) = &referentiePerron($Rit); # PERRON

 $Perrons = $leeg;
 if (!&relevant($S{$Rit})) {
   &Print("Rit $Rit is niet relevant. S{$Rit}:$S{$Rit}.");}
 elsif (&vast($Rit))      {
   $Perrons = &add($P0{$Rit},$Perrons);}
```

```
 else {
  local($p)=$minPerron;
  while ($p<=$maxPerron) {
   if (&valide($p) && &toegestaan($Rit,$p)) {
      $Perrons = &add($p,$Perrons);}
   ++$p;
  }
 }

 if ($Perrons ne $leeg)
 {$Kost = $Leeg;
  $Cond = ' = 1';
  while ($Perrons ne $leeg)
  {($p,$Perrons)    = &remTop($Perrons);
   $Tw              = &toewz($Rit,$p);
   $Tws{$Tw}        = $true;
   $n               = &afstand($RefPerron,$p);
   $s               = &waarde($S{$Rit});
   if ($n != 0) {$n = 10+($n*$s);}
   $Kost            = ' + '.$n.' '.$Tw.$Kost;
   $Cond            = ' + '.$Tw.$Cond;
  }
  $Kosten           = $Kost.'&'.$Kosten;
  $CondsEenPerron   = $Cond.'&'.$CondsEenPerron
 }
}
```

C.4 De functie CheckLengte

Eerst wordt er bepaald, m.b.v. het array $TydDom, op welke tijdstippen er ritten aankomen of vertrekken. Voor elk gevonden tijdstip wordt bepaald, in $NuGeres, welke ritten er op dat moment een reservering hebben lopen.

Vervolgens wordt voor elk perron nagegaan welke van deze ritten erop geplaatst zou kunnen worden. Als de gezamelijke lengte ($Max) van deze ritten die van het perron overschrijdt, dan wordt de string $Lengte.' <= '.$PerronLengte toegevoegd aan de globale variabele $CondsLengte. Het tijdstip $unct wordt als commentaar toegevoegd.

```
sub CheckLengte
{local($t);       # SEC
 local($StartReserv,$EindeReserv,$NuGereserv);
         # RITTEN, RITTEN, RITTEN
 local($p);       # PERRON
 local($Lengte);  # STRING
 local($Max);     # INT

 local(%TydDom);  # $TydDom{SEC} -> BOOL
 local(%algehad); # $algehad{STRING} -> BOOL
```

```
 $CondsLengte   = $leeg; # globale variabele
 $NuGereserv    = $leeg; # globale variabele

 foreach $t (keys %StartReserv) {$TydDom{$t} = $true;}
 foreach $t (keys %EindeReserv) {$TydDom{$t} = $true;}

 foreach $t (sort keys(%TydDom))
 {$StartRes  = $StartReserv{$t};
  $EindeRes  = $EindeReserv{$t};
  $NuGeres   = &conc(&subtr($NuGeres,$EindeRes),$StartRes);
  $p         = $minPerron;
  while ($p <= $maxPerron)
  { ($Lengte,$Max) = &totLengte($p,$NuGeres,$t);
    if ($Max>&perronLengte($p) && !$algehad{$Lengte})
    {$algehad{"$Lengte"} = $true;
     $unct=&uncodeSec($t);
     $CondsLengte = &add("$Lengte <= $PerronLengte\ $unct",$CondsLengte);
    } ++$p;
  }
 }
}

sub totLengte
{local($p,$Ritten,$t)=@_; # PERRON, RITTEN, SEC
 local($Lengte,$Max);     # STRING, INT
 local($Rit);             # RIT
 local($Tw);              # TOEWYZ-VAR
 local($len);             # INT

 if ($Ritten eq $leeg) { ($Lengte,$Max) = ('',0);}
 else
  {
    ($Rit,$Ritten) = &remTop($Ritten);
    ($Lengte,$Max) = &totLengte($p,$Ritten,$t);
    if ($T{$Rit} <= $t && $t < $T{$Rit}+$R{$Rit})
    { $Tw  = &toewz($Rit,$p);
      if (&mogelyk($Tw))
      { $len    = &wagenLengte($Rit);
        $Lengte = ' + '.$len.' '.$Tw.$Lengte;
        $Max    = $Max+$len;
      }
    }
  }
 ($Lengte,$Max)
}
```

C.5 De functie CheckTegelykertyd

Deze functie gaat alle ritten paarsgewijs af, en gaat na of de beide ritten tegelijkertijd op een perron geplaatst mogen worden. Zoniet, dan wordt voor elk perron waarop ze beide geplaatst kunnen worden een conditie opgenomen die voorkomt dat beide ritten tegelijkertijd op dat perron geplaatst worden. In de onderstaande code luidt deze conditie `$Tw0 + $Tw1 <= 1`.

Twee ritten kunnen niet tegelijkertijd op eenzelfde perron geplaatst worden als:

- Hun aankomstintervallen overlappen, of,
- hun vertrekintervallen overlappen, of,
- de één eerder aankomt, maar later vertrekt.

Om het paarsgewijs aflopen eenvoudig te formuleren, maken we gebruik van de lijst notatie van Perl zelf; `List0` wordt geïnitialiseerd op de lijst van alle mogelijke ritten. De operatie `shift List0` haalt de eerste rit uit de lijst, en levert deze af; `List0` wordt dus zelf ook gewijzigd.

```
sub CheckTegelykertyd {

 $CondsNietTegelykerTyd = $leeg;
 @List0                 = sort keys(%P0);
 while (@List0)
 {$Rit0              = shift @List0;
  $StartAankInterv0  = $T{$Rit0};
  $EindeAankInterv0  = $T{$Rit0}+$R{$Rit0}-$H{$Rit0};
  $StartVertrInterv0 = $T{$Rit0}+$H{$Rit0};
  $EindeVertrInterv0 = $T{$Rit0}+$R{$Rit0};
  @List1             = @List0;
  while (@List1)
  {$Rit1 = shift @List1;
    $StartAankInterv1  = $T{$Rit1};
    $EindeAankInterv1  = $T{$Rit1}+$R{$Rit1}-$H{$Rit1};
    $StartVertrInterv1 = $T{$Rit1}+$H{$Rit1};
    $EindeVertrInterv1 = $T{$Rit1}+$R{$Rit1};

   if (  &Overlappend($StartAankInterv0, $EindeAankInterv0,
                      $StartAankInterv1, $EindeAankInterv1)
      || &Overlappend($StartVertrInterv0,$EindeVertrInterv0,
                      $StartVertrInterv1,$EindeVertrInterv1)
      || $StartAankInterv0  < $StartAankInterv1 &&
         $StartVertrInterv0 > $StartVertrInterv1
      || $StartAankInterv0  > $StartAankInterv1  &&
         $StartVertrInterv0 < $StartVertrInterv1
      )
   { $p = $minPerron;
     while ($p <= $maxPerron)
     { ($Tw0,$Tw1) = (&toewz($Rit0,$p),&toewz($Rit1,$p));
       if (&mogelyk($Tw0) && &mogelyk($Tw1))
       { $CondsNietTegelykerTyd =
```

```
            "$CondsNietTegelykerTyd$Tw0 + $Tw1 <= 1&";
        }
        ++$p;
      }
    }
  }
 }
}
```

C.6 De functie RoepCplxAan

Deze functie schrijft eerst de kostenfunctie en de opgestelde condities weg naar de file `f.cplxinput.lp`, waarbij ook de juiste keywords die vereist worden door Cplex, als `"Minimize"` etc., worden tussengevoegd.

Na het keyword `"Integer"` worden al de geïntroduceerde toewijzingsvariabelen opgesomd; Cplex zal al deze variabelen de waarde 1 of 0 toekennen.

Over het algemeen zal de toewijzing waarmee de hoofd-functie `MaakCons` is aangeroepen dicht bij de te construeren optimale oplossing liggen, zoniet ermee samenvallen. De file `f.cplxinput.lp` bevat hierover echter geen enkele informatie, althans niet op een wijze waarop Cplex er iets mee doet. Om Cplex toch een indicatie te geven in welke volgorde de toewijzingsvariabelen op 1 gezet moeten worden, wordt de file `f.cplxinput.ord` opgesteld. Deze file bevat voor elke toewijzingsvariabele een integer, die de prioriteit aangeeft. Cplex zal nu volgens de opgegeven prioriteit de toewijzingsvariabelen op 1 proberen te zetten; op deze wijze wordt de informatie uit de vorige toewijzing gebruikt en zal Cplex meestal sneller tot een oplossing komen.

Cplex moet een NP-compleet probleem oplossen, wat o.a. wil zeggen dat er geen bovengrens is voor de rekentijd. Omdat het voor onze applicatie niet acceptabel is dat er willekeurig lang op gewacht moet worden, wordt er een bovengrens, bijv. 5 seconden, aan de rekentijd gesteld. Dit kan worden aangegeven in de file `cplex.par`, die er dan alsvolgt uitziet:

```
limit       time          5
```

Cplex loopt in een bepaalde volgorde alle mogelijke combinaties af, waarbij de meest optimale gevonden oplossing wordt bijgehouden. Het algoritme stopt als het kan besluiten dat de gevonden oplossing inderdaad de meest optimale is, dan wel als de gestelde rekentijd is afgelopen.

In het laatste geval moet er dan voor zorg gedragen worden, dat binnen de gestelde rekentijd, als het enigszins kan, een acceptabele oplossing gevonden is, hoe weinig optimaal ook.

Merk op dat ook als de optimale oplossing al snel gevonden is, het nog lange tijd kan duren voordat geconcludeerd kan worden dat de gevonden oplossing daadwerkelijk optimaal is.

De volgorde waarin een rit op de perrons wordt geprobeerd te plaatsen is nu als volgt:

- Probeer eerst het toegewezen perron uit de toewijzing waarmee `MaakCons` is aangeroepen (`$P0{$Rit}`),
- Probeer dan het voorkeursperron uit deze toewijzing (`$P1{$Rit}`),
- Probeer dan het reserve-perron. Dit brengt weliswaar hoge kosten met zich mee, maar omdat dit perron over het algemeen vrij is, is het zeer waarschijnlijk dat deze toewijzing acceptabel is.

- Probeer dan de rit telkens één perron verder van z'n voorkeursperron af te plaatsen.

  Voor deze optie geldt weer dat de ritten die als gewijzigd zijn opgegeven door Cplex eerder van hun toegewezen perron zullen worden afgeplaatst.

Uiteindelijk wordt het script `runcplexOrd` aangeroepen. Dit script ziet er bijv. alsvolgt uit:

```
echo "r f.cplxinput.lp\n
      r f.cplxinput.ord\n
      o\n
      set log f.cplxoutput\n dis sol -\n"
      | cplexmip
```

Dit script leest de files `f.cplxinput.lp` en `f.cplxinput.ord`, roept Cplex aan waarbij een logfile wordt bijgehouden in `cplx.log`. De uitvoer zal worden weggeschreven in `f.cplxoutput`. Voor verdere informatie wordt verwezen naar de manual van Cplex.

De uitvoer wordt uitgelezen, als een rit op een ander perron is geplaatst, dan wordt deze wijziging meegenomen in de variabele `$Wyzigingen`, die uiteindelijk door de hoofdfunctie `MaakCons` zal worden teruggestuurd naar de Toolbus.

```
sub RoepCplxAan
{local($Gewyzigd)=@_;

 if (-e "f.cplxinput.lp") { system("rm f.cplxinput.lp");}
 open(CPL,">f.cplxinput.lp");

 print CPL "Minimize\n";
 &PrintTeksts($Kosten);
 print CPL "\n";

 print CPL "Subject To\n";
 &PrintTeksts($CondsEenPerron.
              &breekTeksts($CondsLengte).
              $CondsNietTegelykerTyd);

 print CPL "Integer\n";
 foreach $Tw (sort keys(%Tws)) { print CPL $Tw,"\n";}
 print CPL "End\n";
 close CPL;

 if (-e "f.cplxinput.ord") { system("rm f.cplxinput.ord");}
 open(ORD,">f.cplxinput.ord");
 print ORD "NAME\n";
 foreach $Tw (sort keys(%Tws))
  {($Rit,$p) = &ontleed($Tw);
   $n = $maxPerron - &afstand($p,&referentiePerron($Rit));
```

De variabele `$n` staat voor de prioriteit waarmee de toewijzingsvariabele `$Tw` op 1 gezet zal worden.

```
   if    ($p == $PO{$Rit})       { $n = 2*$maxPerron + 2;}
```

```
   elsif ($p == $P1{$Rit})      { $n = 2*$maxPerron + 1;}
   elsif ($p == $reservePerron) { $n = 2*$maxPerron;}
   elsif (&isIn($Rit,$Gewyzigd)){ $n = $n+$maxPerron;}
   print ORD &spaties(1),"UP",&spaties(1),$Tw,
                 &spaties(36-(4+length($Tw)+length($n))),$n,"\n";
  }
 print ORD "ENDATA\n";
 close ORD;

 if (-e "f.cplxoutput") {system("rm f.cplxoutput");}
 system("runcplexOrd>out");

 open(CPL, "f.cplxoutput");
 $Wyzigingen=$leeg;
 $Geslaagd=$false;
 while(<CPL>)
 {if (/^t(\w*)_(\d*) (\s*) (1.000000)/)
  {($Rit,$p)=($1,$2);
   $Geslaagd = $true;
   if ($P0{$Rit} != $p )
   {
    &Print("$Rit van $P0{$Rit} naar $p.");
    $P0{$Rit}   = $p;
    $Wyzigingen = "$Wyzigingen$Rit:$p&";
   }
  }
 }
 close CPL;
}
```

D De bibliotheek met basisfuncties

```
sub max    {local($n0,$n1)=@_; if ($n0 >= $n1) {$n0} else {$n1} }
sub min    {local($n0,$n1)=@_; if ($n0 <= $n1) {$n0} else {$n1} }

sub valide
{local($Perron)=@_; $minPerron <= $Perron && $Perron<=$maxPerron; }

sub toegestaan
{local($Rit,$Perron)=@_;
 (  ($Route{$Rit} eq $stadRoute   && 1 <= $Perron && $Perron <= 5  )
 || ($Route{$Rit} eq $streekRoute && 6 <= $Perron && $Perron <= 10 )
 || ($Perron == $reservePerron)
 )
}
```

```
sub afstand
{local($Perron0,$Perron1)=@_;
 if (&valide($Perron0) && &valide($Perron1))
 {if    ($Perron0 != $reservePerron && $Perron1 != $reservePerron)
     {if ($Perron0 <= $Perron1 ) {$Perron1-$Perron0;}
          else                   {$Perron0-$Perron1;} }
  elsif ($Perron0 == $reservePerron && $Perron1 == $reservePerron)
        {0;}
  else  {100;}
 }else  {0;}
}

sub perronLengte
{local($Perron)=@_;
 if    (!&valide($Perron))         {0;}
 elsif ($Perron == $reservePerron) {48;}
 elsif ($Perron == 1)              {24;}
 elsif ($Perron == 2)              {36;}
 elsif ($Perron == 3)              {36;}
 elsif ($Perron == 4)              {36;}
 elsif ($Perron == 5)              {36;}
 elsif ($Perron == 6)              {36;}
 elsif ($Perron == 7)              {36;}
 elsif ($Perron == 8)              {36;}
 elsif ($Perron == 9)              {24;}
 elsif ($Perron ==10)              {24;}
 else                              {0;}
}

sub vast
{local($Rit)=@_; $S{$Rit} eq $opPerStatus || &waarde($S{$Rit}) == $opBufWaarde ;}

sub speling
{local($Rit)=@_;1}

sub waarde
{local($Status)=@_;
 if    ($Status eq  $gers1Status)     {$gers1Waarde;}
 elsif ($Status eq  $gers2Status)     {$gers2Waarde;}
 elsif ($Status eq  $vrtrgStatus)     {$vrtrgWaarde;}
 elsif ($Status eq  $gedetStatus)     {$gedetWaarde;}
 elsif ($Status =~ /^opBuffer(\d+)$/) {$opBufWaarde;}
 elsif ($Status eq  $opPerStatus)     {$opPerWaarde;}
 elsif ($Status eq  $vrtrkStatus)     {$vrtrkWaarde;}
 else                                 {0;}
}
```

```
sub relevant
{local($Status)=@_; &waarde($Status)!=0; }

sub referentiePerron
{local($Rit)=@_;
 if (&vast($Rit)) {$P0{$Rit}} else {$P1{$Rit}} ;
}

sub perrons
{local($p0,$p1)=@_;
 local($Perrons)=$leeg;
 while ($p0 <= $p1) {$Perrons = $p0 .'&'. $Perrons; ++$p0; }
 $Perrons
}

sub first
{local($Paar)=@_; local($x,$y)=split(/,/,$Paar); $x; }

sub second
{local($Paar)=@_; local($x,$y)=split(/,/,$Paar); $y; }

sub wagenLengte
{local($Rit)=@_; &lengte(&type($Wagen{$Rit})) }

sub type
{local($Wagen)=@_; 1 }

sub lengte
{local($Type)=@_; 12 }

sub wagenLengte
{local($Rit)=@_; &lengte(&type($Wagen{$Rit})) }

sub toewz
{local($Rit,$Perron)=@_;
 local($s)='t'.$Rit.'_'.$Perron;
 $s;
}

sub ontleed
{local($Tw)=@_;
 $Tw =~ /^t(\w*)_(\d*)$/; ($1,$2)
}
```

Het `aankomstInterval` wordt gebruikt in het random bepalen van de werkelijke aankomsttijd, het wordt hier voor elke rit op 6 minuten gezet.

In deze versie is er maar een `detectiePunt` en wel 1. En elk detectiepunt ligt op 2 minuten afstand van het station, zoals gedefinieerd in de functie `afstandInTyd`.

```
sub aankomstInterval {local($Rit) =@_;     60*6;}
sub detectiePunt     {local($Rit) =@_;        1}
sub afstandInTyd     {local($DetPunt)=@_; 60*2;}

sub remTop
{local($List)=@_; $List =~ /([^&]*)&(.*)/; ($1,$2) }

sub add{local($Item,$List)=@_; "$List$Item&"}

sub rem
{local($Item,$List)=@_;
 local($List1) = $leeg;
 while ($List ne $leeg) {
  ($Item1,$List) = &remTop($List);
  if ($Item1 eq $Item)
   {$List1 = "$List1$List";
    $List=$leeg
   } else
   {$List1 = &add($Item1,$List1);
   }
  } $List1
}

sub subtr
{local($List0,$List1)=@_;
 local($List)='';
 while ($List0 ne $leeg)
 {($Item,$List0) = &remTop($List0);
  if (!isIn($Item,$List1)) {$List = &add($Item,$List);}
 }
 $List
}

sub conc {local($String0,$String1)=@_; "$String0$String1" }

sub isIn
{local($Item,$List)=@_;
 local($Bool)=$false;
 local($Item1);
 while (!($List eq $leeg || $Bool))
 {($Item1,$List) = &remTop($List);
  $Bool = ($Item1 eq $Item);
 }
$Bool
}

sub PrintTeksts
```

```
{local($Teksts)=@_;
 local($Tekst);
 while ($Teksts ne $leeg)
 {($Tekst,$Teksts) = &remTop($Teksts);
  print CPL $Tekst,"\n";
 }
}

sub breekTeksts
{local($Teksts)=@_;
 local($Tekst0,$Teksts0);
 if ($Teksts ne $leeg)
 {($Tekst0,$Teksts0) = &remTop($Teksts);
   $Teksts = breekTekst($Tekst0).breekTeksts($Teksts0);
 } $Teksts;
}

sub breekTekst
{local($Tekst)=@_;
 local($Teksts)='';
 local($Tekst1,$n);
 while ($Tekst ne '')
 {$Tekst1='';
  $n=0;
  while ($n<6 && $Tekst ne '')
  {
    if ($Tekst =~ /\+ ([a-z0-9_\s]+) (\+|\-|=|<=) (.*)$/)
    {$Tekst1 = $Tekst1.' + '.$1;
     $Tekst  = $2.' '.$3;
    }
    elsif ($Tekst =~ /\- ([a-z0-9_\s]+) (\+|\-|=|<=) (.*)$/)
    {$Tekst1 = $Tekst1.' - '.$1;
     $Tekst  = $2.' '.$3;
    }
    elsif ($Tekst =~ /([a-z0-9_\s]+) (\+|\-|=|<=) (.*)$/)
    {$Tekst1 = $Tekst1.$1;
     $Tekst  = $2.' '.$3;
    }
    else
    {$Tekst1 = $Tekst1.' '.$Tekst;
     $Tekst  = '';
    }
    ++$n;
  }
  $Teksts = $Teksts.$Tekst1.'&'.$leeg;
 }
return $Teksts;
```

```
}

sub spaties
{local($n)=@_;
 $s = '';
 while ($n>0) {$s = " $s";$n=$n-1;}
 $s
}

sub vulNulAan
{local($n)=@_; if ($n<10) {$n  = "0$n";} $n }

sub uncodeSec
{local($s)=@_;
 local($u,$m)=(0,0);
 $u = int($s/3600);
 $m = int(($s-$u*3600)/60);
 $s = $s-$u*3600-$m*60;
 $u = &vulNulAan($u);
 $m = &vulNulAan($m);
 $s = &vulNulAan($s);
"$u.$m.$s"
}

sub TydInSec {local($uur,$min,$sec)=@_; 60*(60*$uur+$min)+$sec }

sub MaakTydInSec
{local($Tyd)=@_;local($t)=0;
 if ($Tyd =~/^\[(\d*),(\d*),(\d*)\]$/) {$t = &TydInSec($1,$2,$3);}
 else {&Print("Tydstip $Tyd heeft verkeerd formaat");}
$t
}

sub inInterval {local($t,$s0,$s1)=@_; $s0<=$t && $t<=$s1}
```