WORDT
NIET UITGELEEND

ngen Faculteit der Wiskunde en Natuurwetenschappen

Vakgroep
Informatica

Afstudeerverslag

PlanMate

De geboorte van een softwarepakket ter ondersteuning van de personeelsroostering in de bouwwereld

juni 1995

Ruben Kawele van der Meer
Fivelstraat 22
9751 BG Groningen
telefoon: 050 717352
studentnummer: 0448753
Bestuurlijke Informatica

Rijksuniversiteit Groningen
Bibliotheek
Wiskunde / Informatica / Rekencentrum
Landleven 5
Postbus 800
9700 AV Groningen

RuG

Voorwoord

Dit document is een verslag van het afstudeerproject, dat door mij in het kader van de studie Bestuurlijke Informatica verricht is aan de faculteit der Wiskunde en Natuurwetenschappen en de faculteit Bedrijfskunde. Deze afstudeerstage hield het voortzetten van het project PlanMate in.
Het project PlanMate had tot doel een kennissysteem te ontwikkelen ter ondersteuning van de personeelsroostering in de bouw en bestond uit een haalbaarheidsstudie, een beperkt prototype en enkele daarbij behorende documenten. Na het afsluiten van het project PlanMate, besloten IKS, KPD Automatisering en de Rijks Universiteit Groningen dit voort te zetten in de vorm van PlanMate II.
Tijdens de uitvoering van PlanMate, werkte ik samen met Jan Glaudé, die mij wegwijs maakte in het bedrijfsleven en voor mij de broodnodige informatie uit de roostermakers wist te halen. Gedurende deze periode heb ik veel geleerd over de grote boze wereld om mij heen en heeft het begrip deadline voor mij een andere betekenis gekregen.
Jan Glaudé is ook de reden waarom er binnen dit verslag af en toe over 'ons' en 'wij' gesproken wordt. Ik heb het dan over de gedeeltes van onze stages die overlapten en die we dus samen hebben volbracht.
Ik kan iedereen een stage aanbevelen op het gebied van software engineering, omdat het zeer veelzijdig is en meer mensenkennis vergt dan elke andere richting binnen de informatica. Daarnaast is het het leukst het hele traject te doorlopen zodat je de software langzaam ziet groeien tot een volwaardige applicatie.

Op het moment dat de tweede versie van dit verslag voor u ligt, is het project PlanMate al in een veel verder gevorderd stadium. Het is zelfs al zo ver gevorderd dat er op het moment een andere naam voor moet worden bedacht, omdat de naam PlanMate al bleek te bestaan. De afbeeldingen van het gebruikersinterface zijn van de nieuwste versie genomen, maar ik zal me beperken tot de functionaliteit van de versie 0.1, die aan het eind van de stage gereed was. Om aan te geven wat de nieuwe mogelijkheden zijn, zal ik in sommige hoofdstukken een enkele keer even ingaan op toegevoegde zaken.

Groningen, juni 1995

Ruben Kawele van der Meer

Rijksuniversiteit Groningen
Bibliotheek Informatica / Rekencentrum
Landleven 5
Postbus 800
9700 AV Groningen

Rijksuniversiteit Groningen
Bibliotheek Informatica / Rekenmachinewetenschappen
Landleven 5
Postbus 800
9700 AV Groningen

Samenvatting

In de haalbaarheidsstudie, die voor PlanMate gepleegd werd, bleek dat er in de bouw een grote behoefte bestaat aan een goed ondersteunend, gebruikersvriendelijk personeelsplanningspakket. Ook bleek dat er een beperkte ondersteuning mogelijk moest zijn om met de planner mee te denken. Dit hield in dat het mogelijk moest zijn een kennissysteem te ontwikkelen voor een aantal planningsproblemen.

Daarom is er binnen het project PlanMate II gepoogd deze zaken te verwezenlijken. Het eerste doel van de afstudeeropdracht was om PlanMate zo goed mogelijk op de roostertaak van de roostermakers te laten aansluiten. Dit gebeurde door kennisacquisitie bij de betrokken bedrijven. Na de opdracht werd dit door middel van prototypen door KPD voortgezet. We konden dus spreken van de ontwikkeling van een gebruikersvriendelijke database.

Het kennissysteem is verwezenlijkt door in beginsel de regelstructuur en objectgeörienteerdheid van de expertshell Nexpert [Neuron Data, zie deel 2] te gebruiken bij de kennisstructurering. Daarbij lag vooral de nadruk op de kennis van ploegcontinuïteit en het minimaliseren van reisafstanden. Deze reisafstanden waren door middel van de koppeling aan een reisplanner makkelijk te verkrijgen.
Bij PlanMate was het van belang dat de berekening van de minimale reiskosten zo efficiënt mogelijk uitgevoerd konden worden. Het was nodig een snel algoritme te hebben, omdat we bij grotere bestanden al snel over een zeer grote probleemruimte konden spreken.

Nexpert brengt als ontwikkelshell echter een aantal nadelen met zich mee, die het gebruik minder interessant maken voor het bedrijfsleven:
1. Een kennissysteem wordt een stuk duurder doordat een hardwarekey meegeleverd zal moeten worden. Een hardwarekey is een stukje hardware aan een LPT-poort van de computer waaraan de runtime versie van Nexpert kan controleren of de gebruiker Nexpert mag gebruiken. Zonder deze key zal Nexpert niet werken.
2. Het aanroepen en laten draaien van Nexpert is traag en niet 100% betrouwbaar.

Omdat de uitvoer van Nexpert in een 'tekst'-vorm gegeven kan worden, was het mogelijk deze uitvoer te scannen en via een parser om te zetten naar een hogere programmeertaal. Door de kennisstructurering in de Nexpert-notatie weer te geven, heb ik geprobeerd aan de hand van deze regels een parser te construeren. Wat dan uiteraard ontbreekt is de objectgeörienteerdheid van Nexpert en het ingebouwde redeneermechanisme.
Dit redeneermechanisme en de objectgeörienteerdheid moesten dus in de bewuste programmeertaal geïmplementeerd worden. De gekozen taal was Visual Basic, omdat KPD geïnteresseerd is in kennissystemen en onder MS-Windows programmeert in Visual Basic.
In de bouw wordt voornamelijk backward en forward chaining toegepast, daarom is alleen een aanzet gegeven voor het ontwikkelen van deze redeneermechanismen.

Inleiding

Het project Personeelsplanning Bouwbedrijven (PPB), dat in PlanMate II z'n uiteindelijke implementatie moest vinden, kwam natuurlijk niet zomaar uit de lucht vallen. Er was binnen de bouwwereld vraag gerezen naar een planningsondersteunend produkt als PlanMate waarna er in het kader van Diskus-C al een haalbaarheidsstudie bij het Coöperatief Bouwbedrijf Moes u.a. te Zwolle werd uitgevoerd naar de ontwikkeling van een op kennis gebaseerd roostersysteem.
Diskus-C is een vervolgproject op Diskus, een acroniem voor Dynamic Interactive Scheduling and Knowledge Utilization Systems. Het oorspronkelijke Diskus-projekt was een gezamenlijk onderzoek van enkele universiteiten en het bedrijfsleven naar rooster- en planningsdomeinen.
De C van het vervolgprojekt Diskus-C staat voor Classificatie, waarmee één van de belangrijkste onderzoeksdoelen van dit projekt wordt aangegeven, namelijk het classificeren van rooster- en planningsdomeinen.
In de haalbaarheidstudie van PlanMate I concludeerde J. Oldenkamp dat de twee belangrijkste kenmerken van het verlangde systeem een vriendelijke gebruikersinterface, en de mogelijkheid om met plannen 'mee te denken' zijn.
Vervolgens trok hij de conclusie dat dit inderdaad haalbaar was en intussen heeft de ontwikkeling van een eerste prototype van PlanMate plaatsgevonden in samenwerking met E. Huisman, H. Numan, R. Jorna, en S. Sibum.
PlanMate I resulteerde in een op het SEC-model[1] gefundeerd, ondersteunend systeem voor het invoeren van de vraag naar werknemers bij nieuwe bouwprojekten en het inroosteren van werknemers op nieuwe bouwprojekten.
De mogelijkheden van PlanMate bleven beperkt tot een gebruikersinterface. Het huidige Projekt Personeelsplanning Bouwbedrijven had als doel om tot een volledig operationeel systeem te komen in samenwerking met andere bouwbedrijven en softwarehuis KPD, dat een commerciële versie van PlanMate op de markt wilde brengen.
Het projekt PPB zou een aantal fasen doorlopen om door middel van de kennis van de taak van de roosterexpert uiteindelijk PlanMate II te implementeren. Deze fasen komen allemaal aan bod in dit verslag en geven een goed beeld van de geboorte van de babyversie van PlanMate, die PlanMate II heet.

In deel één is het nodig de situatie te schetsen waarin PlanMate zal moeten gaan werken. We willen een gestructureerd beeld hebben van de omgeving zodat er gericht naar een oplossing gezocht kan worden.

Deel twee introduceert Nexpert als goede omgeving voor het ontwikkelen van kennissystemen en bevat de door mij in Nexpert vastgelegde kennis uit het kennisdomein van de roostermakers, zoals dat door ons gevonden is.

Deel drie beschrijft de programmeeromgeving Visual Basic, met z'n beperkingen en voordelen, en de ontwikkeling van het gebruikersinterface van PlanMate.

Deel vier geeft een aanzet tot de ontwikkeling van een vertaler voor Nexpert uitvoer naar de programmeeromgeving Visual Basic en mogelijke oplossingen voor de implementatie van redeneermechanismen in deze programmeertaal.

[1] Het Schedulings Expertise Centrum-model is ontwikkeld aan de RUG en heeft tot doel roosterapplicaties centraal onder te brengen zodat er in de toekomst bij andere planningsproblemen van gebruik kan worden gemaakt.

Inhoudsopgave

VOORWOORD 3
SAMENVATTING 5
INLEIDING 7
INHOUDSOPGAVE 9

DEEL 1 13

INLEIDING 15

Hoofdstuk één 17
PLANNING EN ONDERSTEUNING 17
Wat is planning? 17
Roosterplanning in organisaties 19
Is personeelsplanning te ondersteunen? 19
Probleemstelling 20
Conclusie 21

Hoofdstuk twee 23
PLANNING BOUWPLAATSPERSONEEL 23
Wat is de voorgeschiedenis? 23
Hoe gaan de roostermakers te werk? 24
Wat was de algemene planningssituatie? 27
Conclusie 28

Hoofdstuk drie 29
DE BEHOEFTEN BIJ DE ROOSTERMAKERS 29
Wat vinden de bouwbedrijven 29
PlanMate en andere planningspakketten 30
Wat moest de functionaliteit van PlanMate II worden? 30
Welke structuur hebben we nodig? 31

DEEL 2 35

INLEIDING 37

Hoofdstuk vier 39
KENNISSYSTEMEN 39
Definities van kennissystemen 39
Kennissystemen op planningsgebied 40
Conclusie 41

Hoofdstuk vijf 43
DE KENNISACQUISITIE 43
Kennisacquisitie in het algemeen 43
De kennisacquisitie bij PlanMate II 43
Conclusie 45

Hoofdstuk zes 47
DE KENNISSTRUCTURERING 47
De opbouw van Nexpert Object 47
De structurering 48

Hoofdstuk zeven 49
CONTROLEUR EN EVALUATOR 49

De controle van de randvoorwaarden 49
De evaluatie van de doelfuncties 50
Conclusie 51

DEEL 3 **53**

INLEIDING **55**

Hoofdstuk acht *57*
GEBRUIKERSINTERFACES **57**
Waarom een gebruikersinterface? 57
Ontwerpprincipes bij gebruikersinterfaces 58
Prototyping van gebruikersinterfaces 59
Conclusie 60

Hoofdstuk negen *61*
VISUAL BASIC **61**
Programmeren als laatste ontwerpfase 61
Waarom Visual Basic? 61
De taal Visual Basic 62
Conclusie 65

Hoofdstuk tien *67*
DE GEBRUIKER EN PLANMATE II **67**
Het hoofdscherm 67
Gegevens invoeren 68
De weergave van informatie in de rapportagemodules 69
Overige zaken van belang 72
Conclusie 72

Hoofdstuk elf *73*
HET PROGRAMMEREN VAN PLANMATE II **73**
Het gebruikersinterface in Visual Basic 73
Datamanagement in Visual Basic 73
Rapporteren in Visual Basic 75
De afstandstabel 76

Hoofdstuk twaalf *79*
HET REKENEN IN HET ROOSTEREN **79**
De algoritmes in de kennisbanken 79
Een stappenplan 80

DEEL 4 **83**

INLEIDING **85**

Hoofdstuk dertien *87*
DE UITVOER VAN NEXPERT **87**
Wat is de uitvoer van Nexpert? 87
Conclusie 88

Hoofdstuk veertien *89*
DE IMPLEMENTATIE IN VB **89**
Hoe brengen we de objecten in Visual Basic? 89
Conclusie 91

Hoofdstuk vijftien *93*
DE OBJECTEN EN HET REDENEREN **93**
Objectgeörienteerdheid 93
Redeneermechanismen 93

Hoofdstuk zestien *95*
HET ALGORITME **95**
De functies en procedures 95

Hoofdstuk zeventien *97*
CONCLUSIES **97**
Oplossingen 97
Laatste opmerkingen 98

BIJLAGE A **99**
Belangrijkste tabellen in PlanMate II 99
BIJLAGE B **101**
Informele regels uit de kennisacquisitie 101
BIJLAGE C **107**
Programmatekst: lijst van vraag op bewerkingen 107
BIJLAGE D **109**
Vertaling algoritme naar Visual Basic 109

LITERATUUR **113**

Deel 1

Personeelsplanning

Inleiding

Bij de ontwikkeling van nieuwe software is het voor degene die de programmatuur moet gaan ontwikkelen een noodzaak zich te verdiepen in de omgeving waarin de software zal gaan draaien. Dit kan gedaan worden door literatuur te raadplegen en door uitgebreide gesprekken te voeren met de betrokkenen op dat vakgebied. Ook het bestuderen van andere software die er op dat bepaalde gebied al bestaat kan echter erg nuttig zijn. Persoonlijk vind ik dit de leukste fase van de software engineering, omdat het de mogelijkheid biedt om allerlei verschillende automatiseringsproblemen te lijf te gaan en er op die manier voor de nodige afwisseling gezorgd kan worden. Vooral de gesprekken met de experts op het gebied van hun expertise zijn erg leerzaam en interessant.
Bij PlanMate II staat een planningsprobleem centraal. Het project heeft de belangstelling van de stichting DISKUS die onderzoekt of er door middel van redeneermechanismen meer geboden kan worden dan alleen rekenkundige en administratieve ondersteuning.
Om ons wat meer te verdiepen in de omgeving waarin gewerkt moest gaan worden was het goed te bekijken wat planing in het algemeen inhoudt en hoe we het in het dagelijks leven tegenkomen. In hoofdstuk één behandel ik ook de planningsproblemen in het bedrijfsleven zodat duidelijker wordt welke problemen we tegen kunnen komen bij het plannen van bouwplaatsmedewerkers.
Hoofdstuk twee is een bijgewerkte samenvatting van de behoefteanalyse die door Jan Glaudé en mij opgesteld werd aan de hand van een aantal gesprekken met roostermakers, zodat we meer zicht kregen op de planningsprocessen in de bouwwereld. Uit deze gesprekken werden ook een aantal behoeften duidelijk, die ik in hoofdstuk drie opgetekend heb.
Opgemerkt kan worden dat ik persoonlijk vindt dat er nog niet genoeg duidelijkheid over bepaalde problemen bestaat en dat het zeer goed mogelijk is dat er daarom nog nieuwe gesprekken met roostermakers zullen moeten plaatsvinden.

Hoofdstuk één

Planning en ondersteuning

Aan het begin van het project PlanMate stond al vast dat het een applicatie moest gaan worden die de personeelsplanning zou kunnen ondersteunen. Jan Glaudé en ik hebben eerst bekeken wat personeelsplanning eigenlijk inhoudt, wat we onder ondersteuning van planning in het algemeen verstaan en wat de verschillende opvattingen hierover zijn.
Daarna was het pas zinvol een paar gerichte vragen als probleemstelling te formuleren, die centraal zouden gaan staan bij de ontwikkeling van PlanMate en die dan ook steeds terug zullen komen binnen dit verslag.

Wat is planning?

Planning algemeen beschouwd
Wanneer we heel algemeen naar planning kijken, kunnen we het omschrijven als het bepalen van de volgorde van handelingen. Iets minder algemeen beschouwd is het begrip planning in twee overlappende gedeeltes op te delen, namelijk:

- Het bepalen van methoden tot opdelen van het probleem
- Het vooruit berekenen van een aantal stappen van een probleemoplossingsprocedure

Een volledige **definitie** zou kunnen luiden:

> *Het gebruik van methoden, die er op gericht zijn om het originele probleem in voldoende onderdelen op te delen, het berekenen van een aantal stappen van een probleemoplossingsprocedure, voordat we er één uitvoeren, en het omgaan met wisselwerkingen tussen de onderdelen wanneer deze optreden tijdens deze procedure, met als doel objecten met eigenschappen onderling af te stemmen waarbij we voldoen aan randvoorwaarden en waarbij één of meerdere doelfuncties worden gerealiseerd.*

Planning als cognitieve activiteit
In ons dagelijks leven zijn we eigenlijk voortdurend bezig met plannen. Als we een doel hebben, moeten we een plan verzinnen om het doel te bereiken. De taakanalyse die we dan uitvoeren vindt plaats door associatie in het geheugen met het doel waarop het van toepassing is. Daarbij moet de planner zich realiseren dat dit plan effect kan hebben op het bereiken van andere doelen en dat hij dus wisselwerkingen tussen die doelen moet kunnen opmerken. Deze wisselwerkingen zullen moeten worden meegenomen in de volgende planningsprocessen. Wanneer een planner dus plannen voor doelen wil genereren en veranderen, moet hij alternatieve scenario's kunnen evalueren en sommige doelen laten vallen om andere te bereiken.
In het voorgaande wordt al vaag duidelijk dat het bij plannen lang niet altijd (en eigenlijk vrijwel nooit) mogelijk is om aan alle wensen te voldoen. Ook zien we dat we door middel van berekeningen, ervaring en intuïtie sneller tot een plan kunnen komen doordat de optimale wisselwerking tussen doelen berekend kan worden of de gevolgen van bepaalde wisselwerkingen al bekend zijn. Hier begint zich langzaam de omgeving af te tekenen waarin een kennissysteem het beste tot z'n recht komt. Een berekenend gedeelte in de vorm van een computer en een ervarings- en intuïtiegedeelte in de vorm van kennisbanken.
Duidelijk is dat het ontwikkelen van een kennissysteem voor het gebruik in het dagelijks leven niet tot nauwelijks nuttig genoemd kan worden door het geringe gehalte aan expertise en berekeningen die we in het algemeen nodig hebben voor alledaagse handelingen.
Wanneer we echter naar complexere planningssituaties gaan kijken komen we al gauw bij roosterplanning in organisaties terecht en wordt het dus interessant om te bestuderen of het ontwikkelen van een kennissysteem voor de bewuste situatie een goede ondersteuning voor de planner kan vormen.

Computerondersteuning bij planning
Wanneer we een doel proberen te bereiken bij het plannen, bewegen we ons door de zoekruimte in een poging een oplossing te bereiken voor dat specifieke probleem. Het zoeken door die zoekruimte kan op een aantal manieren, waarbij die verschillende methodes weer het best tot hun recht komen in bepaalde situaties. Wat belangrijk is, is het onderscheiden van verschillende subproblemen zodat een complexe planningssituatie opgedeeld kan worden in stukjes. Een voordeel van deelproblemen is, dat we op deze manier een moeilijk probleem misschien wel opdelen in een aantal makkelijkere deelproblemen. Het belangrijkste doel daarvan is echter dat we deze deelproblemen zodanig definieren, dat we de staat van de probleemoplossingsprocedure niet voortdurend opnieuw hoeven te bepalen. Dit kan namelijk een zeer tijdrovende en overbodige bezigheid zijn. Daarom moeten de deelproblemen geen of weinig effect op elkaar hebben, waarbij een eventuele interactie apart uitgerekend kan worden [Rich/Knight 1991].

Voordeel van het gebruik van een computer, is dat deze in feite een simulatie is van het probleem zoals dat in de 'echte' wereld bestaat. Op deze manier kan de computer dit doen, zolang het effect op de uitkomst van bepaalde stappen uit te rekenen is, dit doen en hij kan de berekende stappen ook opslaan. In de echte wereld zou iemand al deze mogelijkheden niet kunnen onthouden, en al gauw gaan proberen of bepaalde stappen goed uitvallen. Wanneer deze stappen dan verkeerde gevolgen hebben, is het geen simulatie en is de gecreëerde situatie misschien wel niet meer ongedaan te maken.
Voorwaarde is dus wel dat de ruimte waarin we werken voorspelbaar is. Als dit niet zo is kunnen we op z'n best een verzameling genereren van mogelijke uitkomsten. Deze zouden dan waarschijnlijk nog te ordenen zijn naar de kans van optreden. We moeten in dit geval altijd met de mogelijkheid rekening houden dat we een uitkomst krijgen die we niet verwachten.
Er zijn een aantal manieren waarop we een oplossing voor een planningsprobleem kunnen zoeken. We kunnen één stap vooruitdenken, maar dan spreken we van reactieve systemen. Deze putten op één of andere manier uit een kennisbank met oplossingen zodat ze kunnen reageren op bepaalde acties.
We kunnen ook meerdere stappen vooruitdenken en zo proberen te bepalen welke richting we op moeten gaan, willen we de meeste kans op succes hebben. Hier is de manier waarop we dat doen uiteraard weer van belang, en die hangt weer nauw samen met het specifieke planningsprobleem. Het is daarom nodig een algemene strategie te kiezen voordat we een specifieke zoekprocedure kunnen gaan toepassen. Visser en Jorna [1992] onderscheiden een zoekstrategie van een zoekprocedure in de zin dat algemene zoekstrategieen in algemene termen zijn vervat en zo op een grote klasse van problemen toepasbaar zijn. Zoekprocedures bestaan daarentegen uit uitgewerkte voorschriften die meestal weergegeven worden in zoekbomen of in specifiek geformuleerde toestandsruimtes.

Probleemoplossingsstrategiën
Met een goed gekozen zoekstrategie kunnen we vaak vermijden dat we een nieuw plan op moeten stellen op het moment dat ons plan faalt. Het is dus zaak op een gestructureerde wijze door de probleemruimte te zoeken. We moeten daarom in het begin vaststellen welke zoekstrategieen er gebruikt worden door experts op het vakgebied dat we willen gaan ondersteunen. Er worden verschillende strategieen onderscheiden [Jorna/Simons 1992]. De belangrijkste op het gebied van planning zijn:

- Probleemreductie: Door een opdeling in deelproblemen kunnen we de oplossing per probleem proberen te vinden. We gaan er hierbij vanuit dat we het oorspronkelijke complexe probleem opdelen in makkelijker op te lossen deelproblemen.
- Voorwaarts redeneren: We noemen dit ook wel data-georienteerd zoeken, wat al aangeeft dat we geschikt geachte gegevens combineren om geschikte gevolgtrekkingen te bereiken. Per gevolgtrekking kunnen we dan bekijken of het nut heeft door te gaan.
- Achterwaarts redeneren: Bij achterwaarts redeneren zoeken we voor de te bereiken conclusies geschikte premissen gezocht, waar we per premisse eventueel weer nieuwe premissen moeten zoeken. We noemen dit ook wel doel-georienteerd zoeken, omdat we het doel proberen te bereiken door aan de gestelde voorwaarden te voldoen.
- Constraint satisfaction: Dit komt in het kort neer op het inperken van de probleemruimte, door deze zo expliciet mogelijk te maken. Dit kunnen we doen door een aantal toestanden te selecteren die aan bepaalde voorwaarden voldoen. Deze manier van zoeken komt goed tot z'n recht bij het creeeren van personeelsroosters.

De strategie (of combinatie van strategieen) die gebruikt wordt, is meestal afhankelijk van de strategie die in de praktijk door de domeindeskundigen wordt gebruikt. Wanneer we deze hebben vastgesteld kunnen we de specifieke zoekprocedure gaan vaststellen waarmee we willen gaan zoeken.

Probleemoplossingsprocedures
Bij een specifieker zoekprobleem (zoals bijvoorbeeld het zoeken van een waarde uit een zoekboom) kunnen we een aantal zoekprocedures toepassen. In de literatuur worden een aantal verschillende procedures vermeld. Naast het eenvoudige 'gis-en-mis' noemt men onder andere horizontaal zoeken (alle mogelijkheden één voor één nagaan), verticaal zoeken (per mogelijkheid doorzoeken tot een dood spoor gevonden is) en optimaal zoeken (het met een evaluatiefunctie evalueren van de resultaten van de bereikte toestand waarna met het meest optimale resultaat verder wordt gegaan).
Het systeem dat aan het plannen is, moet wel een aantal mechanismen bezitten. [Rich/Knight 1991] onderscheidden de volgende mechanismen:

- Een mechanisme om een oplossing te herkennen.
  Zonder dit mechanisme weten we niet dat we een oplossing hebben, en kunnen we blijven zoeken.
- Een mechanisme om een 'dood spoor' te herkennen.
  Dit voorkomt dat we door blijven rekenen met een plan dat al lang niet meer kan slagen.
- Een mechanisme om een 'bijna goede oplossing' te herstellen.
  Wanneer we ontdekken dat we op een dood spoor zijn beland is het mogelijk terug te gaan naar de vorige oplossing om daar een andere uitkomst te proberen. Dit proces van terugkoppeling wordt ook wel backtracking genoemd. Daarom is het ook belangrijk dat eventuele interacties die al plaatsgevonden hebben weer ongedaan gemaakt kunnen worden. We houden op deze manier het effect van de verschillende stappen op ons plan lokaal, zodat bij het falen van één stap, niet direct het hele plan mislukt is.

Op deze manier moet het mogelijk zijn gestructureerd door een planningsprobleem heen te zoeken. Bij het ontwikkelen van een kennissysteem is het belangrijk de startegieen en procedures te herkennen, die gebruikt worden. Op deze manier kan de planning vertaald worden in kennisbanken.

Roosterplanning in organisaties

In organisaties komt roosterplanning al snel neer op het optimaal benutten en onderling op elkaar afstemmen van schaarse middelen. Dit afstemmen is meestal een complexe aangelegenheid omdat niet alleen technische, maar ook politieke zaken een rol spelen. Ben Jansen [1985] onderscheidt een viertal kriteria die volgens hem gehanteerd moeten worden bij de vormgeving van een rooster in bedrijven:

- exteme kriteria
- interne, bedrijfskundige kriteria
- persoonsgebonden kriteria
- psycho-somatische en psycho-sociale kriteria

Deze kriteria zal ik even vluchtig verduidelijken.

Externe kriteria
De exteme kriteria hebben te maken met de eisen die van buitenaf aan de organisatie worden gesteld. We praten dan over economische, technologische, arbeidsrechtelijke, infrastrukturele en exteme politieke aspecten. We zien hier al dat de vormgeving van een arbeidsrooster een ingewikkelde zaak kan zijn.

Interne, bedrijfskundige kriteria
Hier hebben we direct of indirekt te maken met de efficiëntie van de organisatie. Kijken we naar de regeling van de werktijden binnen een organisatie dan spelen produktie-technische, taakstructurele, personele, arbeidsvoorwaardelijke, interne infrastructurele en interne politieke zaken een rol. Ook op dit niveau is er al een grote mate van complexiteit en is het belangrijk deze in de hand te houden, maar duidelijk is dat een goede planning kostenbesparend is.

Persoonsgebonden kriteria
De problemen, die personeelsleden ervaren tengevolge van het werken met een bepaald rooster, en de behoeften, wensen en voorkeuren ten aanzien van veranderingen, die daardoor ontstaan, komen natuurlijk altijd naar boven. Wanneer men persoonsgebonden kriteria wil toepassen onderscheiden we het kwantitatieve aspect en het kwalitatieve aspect.
We kunnen opmerken dat het om verschillende redenen niet zo simpel hoeft te zijn om aan de eisen van personeelsleden tegemoet te komen. We mogen dus niet te snel concluderen dat er helemaal niets met de wensen van de werknemers wordt gedaan.

Psycho-somatische en psycho-sociale kriteria
Deze kriteria zijn gebaseerd op wetenschappelijke kennis over de mogelijke invloed die de verschillende roosterkenmerken in hun onderlinge samenhang kunnen hebben op het lichamelijke en sociale welzijn van het personeel (bijvoorbeeld stress).

Is personeelsplanning te ondersteunen?

Iedere personeelsplanner zal in eerste instantie op deze vraag antwoorden dat dit niet mogelijk is. De roostertaak is te ingewikkeld en bevat teveel kennis van het werknemersbestand om volledig ondersteund te worden. Pas wanneer de roostermaker zich realiseert dat ondersteuning ook kan betekenen dat de roostertaak een stuk gemakkelijker wordt, zal hij onderkennen dat die ondersteuning misschien toch wel handig is. Het blijft dan echter de vraag of de huidige technologie de vaak uitgebreide planningen aankan.

Snel werken met grote bestanden
Wanneer we kijken naar de roostertaak waar een personeelsplanner mee te maken heeft, zien we dat hij simpelweg een koppeling maakt tussen werk en werknemers. Deze koppeling is bij kleine bestanden zeer goed te maken door een personeelsplanner, omdat de hoeveelheid informatie nog goed te bevatten is. Zodra de bestanden echter groter worden, wordt de taak steeds ingewikkelder en gaat de ervaring van de roostermaker een steeds grotere rol spelen.
Tot voor kort konden betaalbare computers dit nog niet ondersteunen. Het werken met grote databestanden en het leggen van relaties tussen objecten, zonder daarbij gedefinieerde regels te overtreden, kostte storend veel tijd. Daarnaast was de resolutie van de VGA monitor vaak niet voldoende om een overzichtelijk beeld te geven van alle beschikbare gegevens. Dit was niet alleen vervelend om mee te werken, maar kwam de snelheid van het planningsproces ook niet ten goede. De ondersteuning bleef op die manier beperkt tot controle van het rooster, het opslaan van de roosters en een duidelijke, overzichtelijke uitdraai van gegevens.
Tegenwoordig ontwikkelt de computerindustrie in hoog tempo nieuwe processoren en worden de kloksnelheden voortdurend hoger. Daardoor wordt de personeelsplanning geleidelijk aan een gebied dat door personal computers

ondersteund kan gaan worden en zien we dat verschillende bedrijven deze markt proberen te veroveren. Personeelsplanning is dus op de huidige markt nog maar mondjesmaat ondersteund door middel van computersoftware, maar lijkt een grote toekomst te hebben.

Is er vraag naar planningsondersteuning?
Deze vraag dient eigenlijk per bedrijfstak onderzocht te worden, wanneer we naar een ondersteuning willen die verder reikt dan het implementeren van een elektronisch planbord. Dit laatste zal bij iedere planningssituatie een goede ondersteuning kunnen zijn, al is het alleen maar omdat er dan een eenduidige personeelsplanning binnen een bedrijf aanwezig is.
Over de gehele linie kunnen we stellen dat elk bedrijf geïnteresseerd is in software die het samenstellen van een efficiënt rooster mogelijk maakt. Een betere indeling van de factor arbeid heeft een hogere produktiviteit tot gevolg en dat betekent weer meer omzet met dezelfde arbeidskosten.
Jan Glaudé en ik hebben dit vraagstuk binnen de bouwwereld onderzocht en onze bevindingen opgetekend in het informele behoeftemodel. In het volgende hoofdstuk zal ik de belangrijkste bevindingen daaruit nog even kort herhalen. Duidelijk werd in ieder geval dat bij de roostering binnen de bouwwereld de vraag wel degelijk bleek te bestaan, en er nog geen software op de markt is die aan deze vraag kan voldoen.

Probleemstelling

Uit de voorgaande paragrafen blijkt dat er in het algemeen wel degelijk een basis lijkt te bestaan voor softwarematige ondersteuning van personeelsplanningen. De huidige planningspakketten zijn nog beperkt en ondersteunen nog niet optimaal. Het is nu zinvol een aantal vragen centraal te stellen in de verdere ontwikkeling van PlanMate. Deze vragen zijn uiteraard door Jan Glaudé en mij opgesteld na de gesprekken met alle betrokken partijen, waardoor er een beeld ontstond wat de functionaliteit van PlanMate moest gaan worden.
De probleemstelling luidde als volgt:

1. Is er vraag naar ondersteuning voor de personeelsroostering binnen de bouw, en wordt deze ondersteuning op dit moment al op een softwarematige manier ondersteund?
   Dit was uiteraard een vraag waarbij de commerciele kant van PlanMate liever zag dat er nog geen ondersteuning beschikbaar is, terwijl de wetenschappelijke kant daar weinig tot geen boodschap aan had.
2. Wat kan de rol zijn van de eerste versie van PlanMate, die door de RUG in samenwerking met bouwbedrijf MOES werd ontwikkeld?
   Hier was het voornamelijk de vraag of we door moesten borduren op de al ontwikkelde software, of dat we er van moesten leren en het beter zou zijn een 'nieuw' begin te maken.
3. Hoe passen we PlanMate in de bestaande planningssituaties binnen de bouwbedrijven, en hoe maken we de applicatie zo generiek mogelijk?
   Hier kwam de commerciele opzet van PlanMate weer naar boven. Het ontwikkelen van een systeem dat in zoveel mogelijk bedrijven kan functioneren zou op een grotere markt kunnen worden afgezet. Dit neemt niet weg dat het een interessante bijkomstigheid was in de ontwerpfase.
4. Hoe gaan de roostermakers te werk en op welke punten en op welke manier willen ze ondersteund worden?
   Deze vraag was voornamelijk van belang op het moment dat we naar de ontwikkeling van het gebruikersinterface keken, maar de handelwijze van de roostermakers was ook nuttig bij de ontwikkeling van de kennisbanken.
5. Kan er een (beperkt) kennissysteem ontwikkeld worden voor deze vorm van personeelsroostering, en welke kennis komt daarvoor het meest in aanmerking?
   Dit was voornamelijk de vraag die Jan Glaudé moest gaan beantwoorden.
6. Hoe ontwikkelen we een gebruikersinterface waarmee de vaak onervaren, maar ook de ervaren gebruikers overweg kunnen?
   Een belangrijk onderwerp binnen deze stage. Software moet werkbaar zijn voor de gebruiker, want als die er niet mee kan werken, is er geen sprake van ondersteuning.
7. Is het mogelijk om Nexpert, als ontwikkelomgeving voor kennisbanken, te gebruiken en de uiteindelijke implementatie in een krachtigere moderne programmeertaal te laten plaatsvinden door middel van een compiler?
   Een compiler is, commercieel gezien, interessant doordat het een stuk voordeliger is. Voor de ontwikkelfase van de kennisbanken is het echter ook nuttig doordat we nu wel de expertshell kunnen gebruiken. Dit vereenvoudigt namelijk de ontwikkeling aanzienlijk.

Gedurende het hele ontwikkeltraject van PlanMate zijn er antwoorden gevonden op deze en vele andere vragen. Ik heb in dit verslag geprobeerd uit te leggen waarom bepaalde beslissingen gemaakt zijn en waarom op andere vragen nog moeilijk antwoord te geven was.
Daarbij heb ik voornamelijk de aandacht gericht op de ondersteuning van de planning in de vorm van het gebruikersinterface. Dat is namelijk de enige plek waar de gebruiker communiceert met de software, en het bepaalt dus bijna volledig de indruk die gebruikers van het pakket krijgen.
Daarnaast heeft de ontwikkeling van de kennisbanken centraal gestaan binnen onze stage.

Pas als de commerciele versie van PlanMate de markt op gaat, zal echter blijken of de gevonden antwoorden en de gemaakte beslissingen de juiste waren.

Conclusie

We zien dat bij een algemeen planningsprobleem al een hoeveelheid zaken een rol spelen die moeilijk vast te leggen zijn en op intuïtie, ervaring en mensenkennis van de roostermaker aankomen. Bij het verkennen van de kennis die bij de roostermakers aanwezig was binnen PlanMate, zien we dat we vooral met de externe kriteria en de interne, bedrijfskundige kriteria te maken hebben wanneer we PlanMate willen plaatsen in het roosterproces in de bouwwereld.
De huidige softwaremarkt voorziet niet in de vraag naar planningsapplicaties, omdat een planningsapplicatie veel gebruik maakt van databases en selecties op databases. Daarom is snelle hardware een vereiste. De huidige technologie begint aan die eisen te voldoen, waardoor er een markt voor planningsondersteuning met behulp van computertoepassingen ontstaat. PlanMate moest voor deze markt binnen de bouwwereld ontwikkeld worden.

Hoofdstuk twee

Planning bouwplaatspersoneel

Voordat we kunnen gaan kijken of er in de bouw vraag is naar ondersteuning van personeelsplanning, is het nuttig eerst te kijken wat deze specifieke planning precies inhoudt.
Hiervoor is het nodig dat de structuur binnen het bouwbedrijf goed bekend is. In de eerste fase van het project PPB is deze modellering uitgevoerd. In de volgende paragrafen zal een korte samenvatting van de meest relevante zaken, die bij deze modellering duidelijk werden, gegeven worden. Daarna zal ik weergeven wat de kennisacquisitie, die Jan Glaudé en ik uitvoerden, aan informatie opleverde. Dit is een verhaal dat grotendeels terug te vinden is in de al eerder verschenen documenten van ons.

Wat is de voorgeschiedenis?

PlanMate I
Het project PlanMate I ontstond nadat in het begin van 1992 bouwbedrijf Moes te Zwolle de stichting Diskus benaderde om de mogelijkheid te onderzoeken of het haalbaar zou zijn het plannen van het bouwplaatspersoneel te ondersteunen met een computerprogramma. Hierbij was vooral de vraag interessant of dit programma ondersteund zou kunnen worden door kennistechnologie.
Johan Oldenkamp, Han Numan en Erik Huisman voerden dit onderzoek uit wat resulteerde in een aantal documenten, waarvan 'haalbaarheidsstudie naar de mogelijkheid om de roostering met behulp van een kennissysteem te ondersteunen' [Oldenkamp 1992] en 'kennisacquisitie ter analyse van de ondersteunde taak' [Oldenkamp 1993] voor mij de meest nuttige informatie bevatten. Het protoype dat ontwikkeld werd, is met veel moeite, in juni 1994 geïnstalleerd bij bouwbedrijf Moes. Dit prototype bevatte nog geen kenniselementen en kan het best omschreven worden als een rooster-editor (zie blz. 40). Mede door de moeizame installatie van PlanMate I en de introductie van het eerste prototype van PlanMate II in september 1994 is er vrijwel geen feedback te verwachten over het functioneren van PlanMate I.

Wat kunnen we leren uit PlanMate I ?
In het project Planmate I is er natuurlijk al een taakanalyse uitgevoerd, waardoor bij het begin van PlanMate II al een aardig beeld gevormd kon worden over de manier van roosteren die in de bouw wordt toegepast. We kunnen deze ruwweg in drie verschillende fasen indelen:

- De **lange termijn planning**: deze behelst het invullen van de verworven projecten in de tijd. Deze planning is van belang voor de omzetplanning en dus voor de continuïteit van het bouwbedrijf. Deze planning kent geen tijdshorizon.
- De **korte termijn planning**: hier hebben we het over het toekennen van bouwplaatspersoneel aan bewerkingen waarop de specifieke kwaliteiten van de bouwplaatsmedewerkers gevraagd zijn. We spreken hier over een termijn van 4 tot 8 weken.
- De **hele korte termijn planning**: het moeten vervangen van plotseling uitgevallen werknemers, of het aanvullen van ploegen om bewerkingen op tijd op te leveren kunnen we rekenen onder de hele korte termijn planning. Dit is dus duidelijk een planning op dagniveau.

Uit PlanMate I leren we dat het mogelijk moet zijn de korte termijnplanning met de computer te ondersteunen, waarbij ik persoonlijk het prototype van PlanMate I in verhouding tot de korte termijn planning teveel gericht vindt op de lange termijn planning. De reden hiervan ligt echter in het feit dat PlanMate I alleen op bouwbedrijf Moes gericht was, terwijl in een generieker systeem al snel blijkt dat Planmate I te beperkt is in het weergeven van bewerkingen en dus tekort schiet bij de korte termijn planning.
Door de moeilijkheden rond het installeren van het prototype PlanMate I werden Jan en ik niet toegelaten tot het planningsoverleg bij Moes. Het was zelfs nodig om regelmatig te benadrukken dat wij niets te maken hadden met de moeilijkheden rond PlanMate I, zodat dit geen invloed zou hebben op de sfeer tijdens het plegen van de kennisacquisitie. Het installeren van PlanMate II was daarentegen een prettige gebeurtenis waarbij duidelijk te merken viel dat men het zeer op prijs stelde dat men betrokken bleef bij het project en dat er dus wat voor teruggezien werd uit de gesprekken die gevoerd waren. Daarbij was het helemaal geen probleem dat dit maar een beperkt werkend systeem was en dat de afgesproken datum niet gehaald werd. Ik denk dat hieruit geleerd moet worden dat de klant graag betrokken wil blijven bij een project, en dat wanneer we over een commercieel produkt willen gaan praten uiteindelijk de klant toch koning is.
De zaken die verder uit PlanMate I te halen zijn, komen bij onze eigen taak- en behoefteanalyse ook weer naar voren. Deze zal ik in de volgende paragrafen behandelen.

Wat was de rol van PlanMate I ?
De rol van PlanMate I bij dit project bleef zeer beperkt . Door de keuze van een aantal vaste waarden binnen bouwbedrijf Moes was het moeilijk een generiek systeem te maken van deze versie. Ook programmeertechnisch gezien was het verstandig om het gebruikersinterface opnieuw te programmeren in Visual Basic, omdat dit door KPD gebruikt wordt en het uiterlijk van PlanMate dan meer op dat van PlanCalc[1] zal gaan lijken. Daarnaast was het voor KPD wenselijk voor de verdere ontwikkeling van PlanMate te kunnen werken in hun eigen programmeeromgeving, maar dit werd niet geeist door KPD. Uiteraard diende PlanMate I wel als inspiratiebron bij de ontwikkeling van PlanMate II

PlanMate II
We konden PlanMate I van PlanMate II op de volgende punten onderscheiden:
- PlanMate I werd ontwikkeld door andere personen.
- PlanMate II had tot doel een commercieel produkt te ontwikkelen, terwijl dat bij PlanMate I geen prioriteit had.
- PlanMate II had daarom tot doel het personeelsplanningsprobleem in het algemeen te ondersteunen, terwijl PlanMate I alleen de situatie bij Moes ondersteunde.
- PlanMate II moest ingepast worden tussen de bestaande software, die bij de bouwbedrijven gebruikt wordt.

De andere punten van verschil vinden we terug in de doelstellingen van PlanMate II.
De partners in het project PlanMate II zijn KPD Automatisering, dat het produkt op de markt wil gaan brengen en de ontwikkelomgeving ter beschikking stelde, het Instituut voor Kennissystemen (Jan Glaudé voor de kennisacquisitie), Blaauw Concultancy (Alfred Blaauw als projectmanager) en de RijksUniversiteit Groningen (Ruben van der Meer voor de implementatie en René Jorna en Jan Jongejan voor hun wijsheid).

Wat waren de doelstellingen van het project PlanMate II ?
De belangrijkste doelstelling van PlanMate II was uiteraard dat het perfect moest aansluiten op de wensen van de personeelsroosteraars in de bouw, die het produkt moeten gaan gebruik, en de optimalisatie van het personeelsrooster door het terugdringen van de personeelskosten. Dit terugdringen van de personeelskosten is een ruim begrip en kan onderverdeeld worden in een aantal subdoelen, die gedurende het project PlanMate II steeds meer vorm zou krijgen:
- Het tijdig signaleren van dalen in de personeelsbehoefte en de daarmee samenhangende opzegtermijnen, die door de werkgever in acht genomen dienen te worden.
- Het minimaliseren van de reiskosten die gemaakt worden door de werknemers, wanneer zij een bewerking proberen te bereiken, door een in dit opzicht optimaal rooster te genereren. Opgemerkt dient te worden dat de roostermaker altijd het laatste woord heeft over het rooster.
- Het verbeteren van de administratiemogelijkheden van de personeelsroosteraars, door middel van een overzichtelijk gebruikersinterface, een aantal belangrijke rapportage-mogelijkheden en een koppeling aan andere administratieve software binnen het bouwbedrijf.

Hoe gaan de roostermakers te werk?

Het planningsproces in het kort
De planningsprocessen binnen de verschillende bouwbedrijven blijken veel overeenkomsten te vertonen. Uit het document van Blaauw [1993] kunnen we drie stappen halen zoals we die in het algemeen tegenkomen bij bouwbedrijven:
- Bepaal de behoefte naar werknemers op de verschillende projecten
- Vergelijk deze vraag met het werknemersaanbod van de vaste werknemers
- Rooster de werknemers in op de projecten volgens het behoefteplan

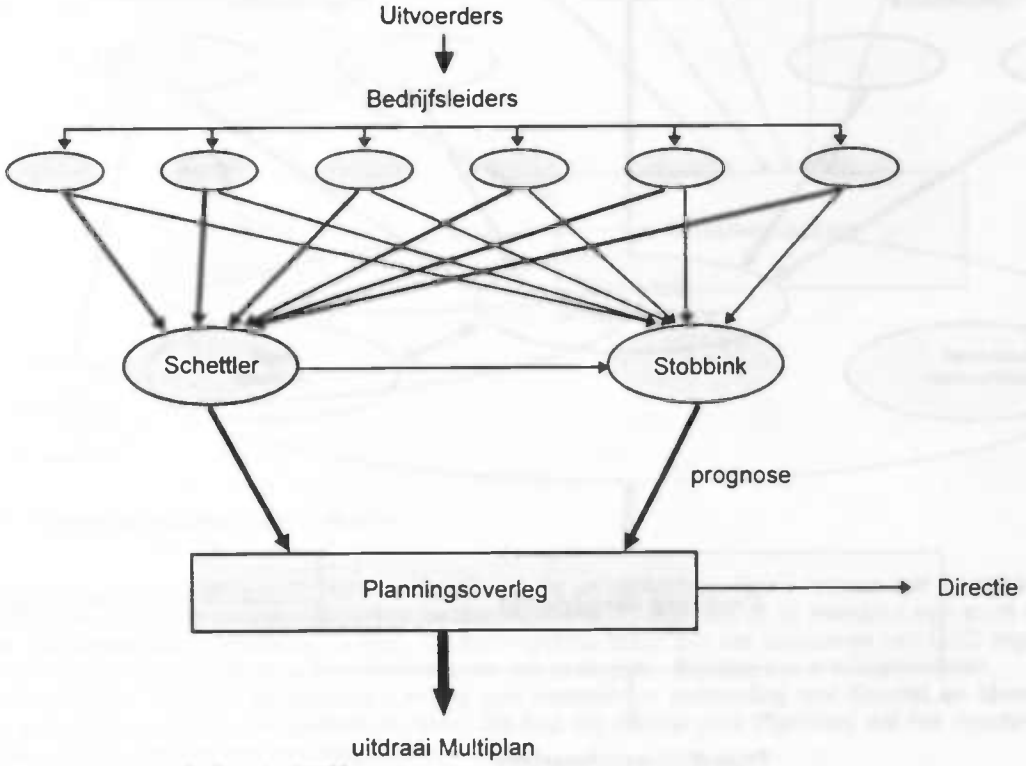
Deze volgorde van stappen is zeer belangrijk voor de opbouw van PlanMate II en we zullen dit proces dan ook nog vaak tegenkomen in de rest van dit verslag.

Het planningsproces bij Moes
Er zijn door ons bij Moes enkele gesprekken geweest met Dhr. Stobbink, Mevr. Schettler en Dhr. Jurjens. Omdat Mevr. Schettler de personeelsplanner is bij bouwbedrijf Moes, ging onze interesse uit naar de werkzaamheden die zij op plangebied verricht. We kunnen de taak van Mevr. Schettler als volgt omschrijven:
- De bedrijfsleiders krijgen een uitdraai uit Multiplan met daarop de planning van de komende weken, zoals die vastgesteld was op de voorgaande planningsvergadering.
- De bedrijfsleiders geven hierop aan wat er op een project veranderd is. Daarbij moeten we denken aan de personeelsbehoeften, de duur van het project en eventueel speciale personeelswensen. In overleg met andere bedrijfsleiders zijn onderlinge personeelsruilen vaak al afgesproken. Ze geven dit aan door op de uitdraaien strepen en namen te zetten. Ieder heeft daar zo z'n eigen manier voor.

[1] PlanCalc is het eerste produkt van KPD dat met Visual Basic is geprogrammeerd. Het is een projectplanningspakket dat naast de 'Windows' versie ook nog onder DOS te verkrijgen is.

- Schettler krijgt deze uitdraaien terug en verwerkt ze puur administratief en past ook een aantal rekentaken toe zoals bijvoorbeeld het signaleren van ontslagtermijnen, ziektetermijnen enz.
- Zij maakt een vraag en aanbodlijst door een streep te zetten na 8 weken en alleen te kijken hoe de situatie daarvoor is.
- In de planningsvergadering wordt gekeken of deze vraag en aanbodlijsten op elkaar aansluiten en de knelpunten worden besproken en opgelost.

Dit is dus in grote lijnen het planningsproces dat plaatsvindt bij Moes. Deze is van belang wanneer we willen gaan bekijken waar PlanMate II precies geplaatst moet gaan worden.



Figuur 1 : Planningssituatie Bouwbedrijf Moes

Het bovenstaande figuur is een schematische weergave van de planningssituatie zoals die bij Moes is. Wanneer we deze naast de (volgens ons) algemene planningssituatie leggen zien we maar weinig verschillen (zie figuur 4).

Het planningsproces bij Grootel

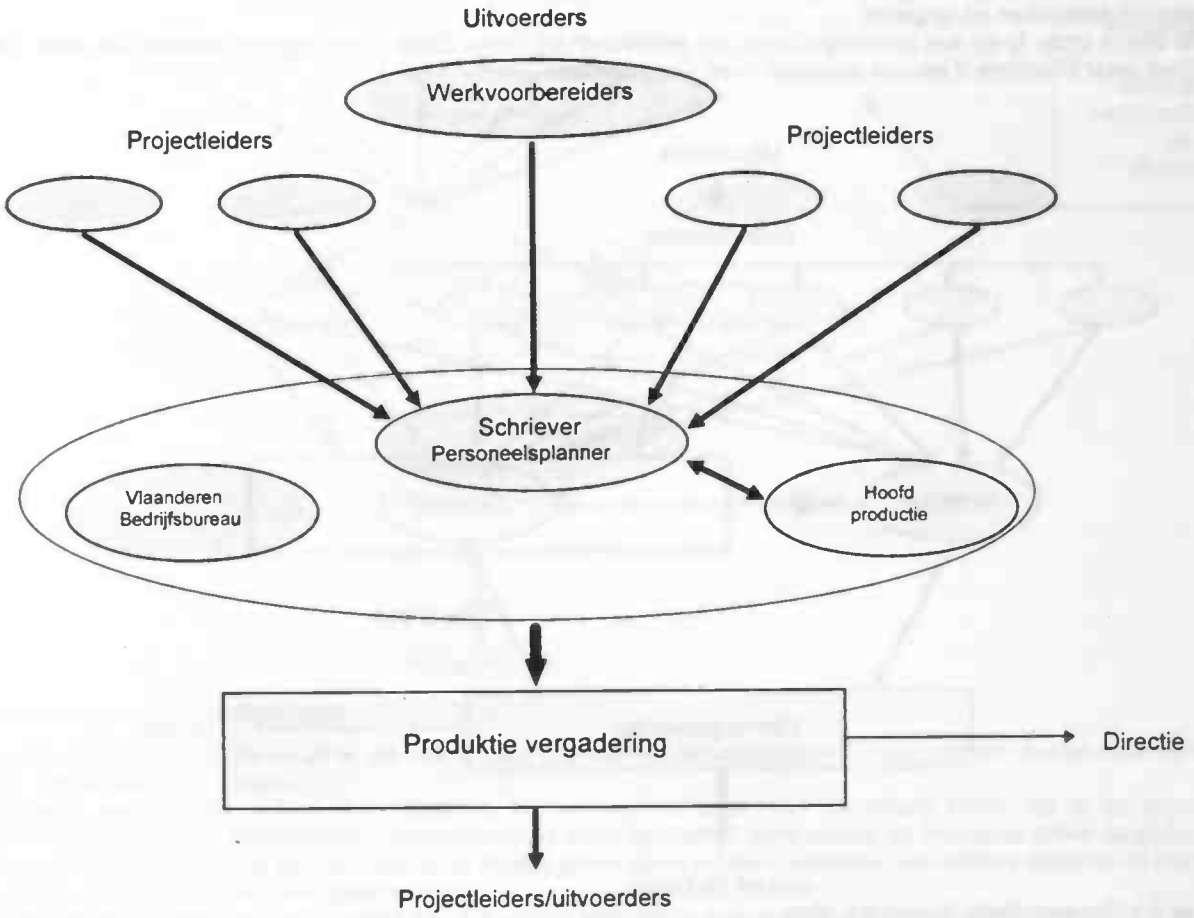
Grootel is opgedeeld in vier delen, die over het midden en het zuiden van Nederland verspreid zijn. Wij bezochten Grootel Midden in Amsterdam en het kantoor in Maarssen.

Binnen Grootel regio Midden heeft de heer Schriever onder andere als taken: het plannen van de lange termijn en het samenstellen van de ploegen waarbij gekeken wordt naar de woonplaats, vaardigheden, het bezitten van een auto en meer van dat soort zaken.

We kunnen zeggen dat Schriever een administratieve taak heeft, maar ook tellingen uitvoert. Zijn taak hield de volgende, voor ons van belang zijnde, werkzaamheden in:

- Het van een man bijhouden wat zijn uurloon en percentage van werken zijn. (De projectleider is financieel verantwoordelijk voor het project en is dus geïnteresseerd in de kosten van een man).
- Het afvlakken van de pieken en dalen in de personeelsbehoefte, die door PlanCalc aangegeven worden.
- Het samenstellen van ploegen, hoewel de uitvoerder voor het grootste gedeelte bepaalt wat hij nodig heeft. Personeelszaken bepaalt waar de werknemers uiteindelijk aan het werk gaan, maar er wordt uiteraard rekening gehouden met wie iedereen goed overweg kan. Er is duidelijk veel overleg voordat beslissingen genomen worden. De ploegsamenstelling wordt ook vaak aan de hand van economische motieven gedaan: bv het indelen van de auto's (er hoeven niet altijd dezelfde disciplines in dezelfde auto te zitten).
- Het rekening houden met individuele wensen van werknemers (interne omscholing en cursussen zijn mogelijk, mits in het belang van het bedrijf).

Elke vier weken zijn alle partijen aanwezig op een **produktievergadering** waarin:
- de toekenning van personeel aan werk wordt vastgesteld.
- wordt geschat hoeveel taakuren een bepaalde taak krijgt (dit is eventueel tijdens de uitvoering, op aangeven van de uitvoerder, te veranderen)

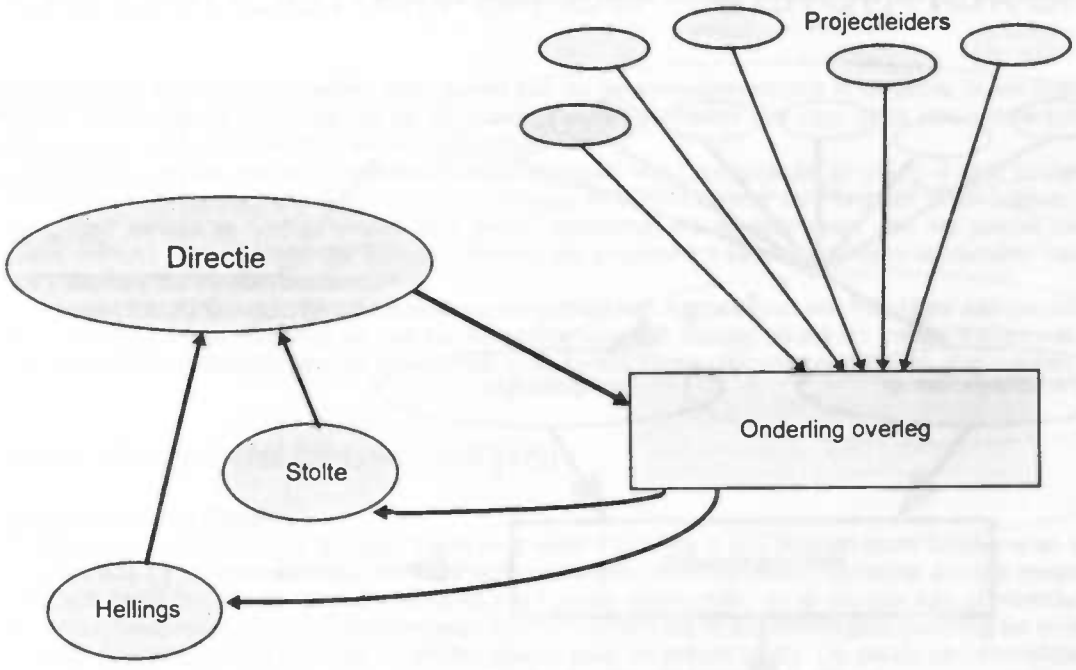


Figuur 2 : Planningssituatie bouwbedrijf Grootel

De overeenkomsten met Moes zijn groot. Wij zelf zagen eigenlijk in Jan Schriever en Mevr. Schettler de personen die precies doen, wat PlanMate ook zou moeten gaan doen, en dat was ook de mening van Rens ten Hagen, die een belangrijke functie vervult op de automatiseringsafdeling van Grootel in Eindhoven.

Het planningsproces bij Hendriks
Bij Hendriks werd gepraat met Hellings, Stolte en vd Wetering. Hellings is van het bedrijfsbureau in Den Bosch, Stolte is van het bedrijfsbureau in Oss en vd Wetering is een projectleider in diezelfde plaats. Ze gebruikten hier PlanCalc en Lotus om hun planning zo goed mogelijk te laten verlopen. Net als bij de andere bouwbedrijven hadden we bij Hendriks te maken met de projectleiders, die wat dichter op de werkvloer zitten dan het bedrijfsbureau en waar dus ook een groot gedeelte van de zeer korte-termijn planning wordt gedaan. PlanCalc wordt eenmalig gebruikt voor de planning op de lange termijn. Ook wordt hiermee een prognose gemaakt en omdat alle werken in PlanCalc terug te vinden zijn, geeft dit een goed overzicht. Deze prognose wordt naast het eindresultaat gelegd om te kijken of hij voldaan heeft. Van de mogelijkheden om in PlanCalc personeel aan de werken te verbinden en om het tempo van deze mensen bij te houden, wordt geen gebruik gemaakt. Het aantal relaties wordt beperkt tot één: de start/start relatie.
Het bedrijfsbureau maakt de planning, maar het toewijzen van personeel gebeurt in gezamenlijk overleg door bedrijfsleider en projectleiders. De planning vindt plaats op grond van 100% gezonde, vakbekwame bouwplaatsmedewerkers. Als een bouwplaatsmedewerker toch 'mislukt', dan verzinnen de projectleiders iets anders. De taak van Stolte en Hellings kan als volgt onderverdeeld worden: Het bijhouden van de uurkosten per bouwplaatsmedewerker en doorgeven van deze gegevens aan de projectleider. Het maken van een calculatie aan het begin van een project en tussentijdse projectbewaking. Het signaleren van pieken en dalen in de personeelsbehoefte. Het controleren van de produktiviteit op langere termijn. Het signaleren van discontinuïteit voor de uren: bestede uren nemen toe, maar de produktie

niet en wekelijks overleggen op het bedrijfsbureau om te proberen personen of ploegen, die overschieten, over te plaatsen naar andere projecten.



Figuur 3 : Planningssituatie bouwbedrijf Hendriks

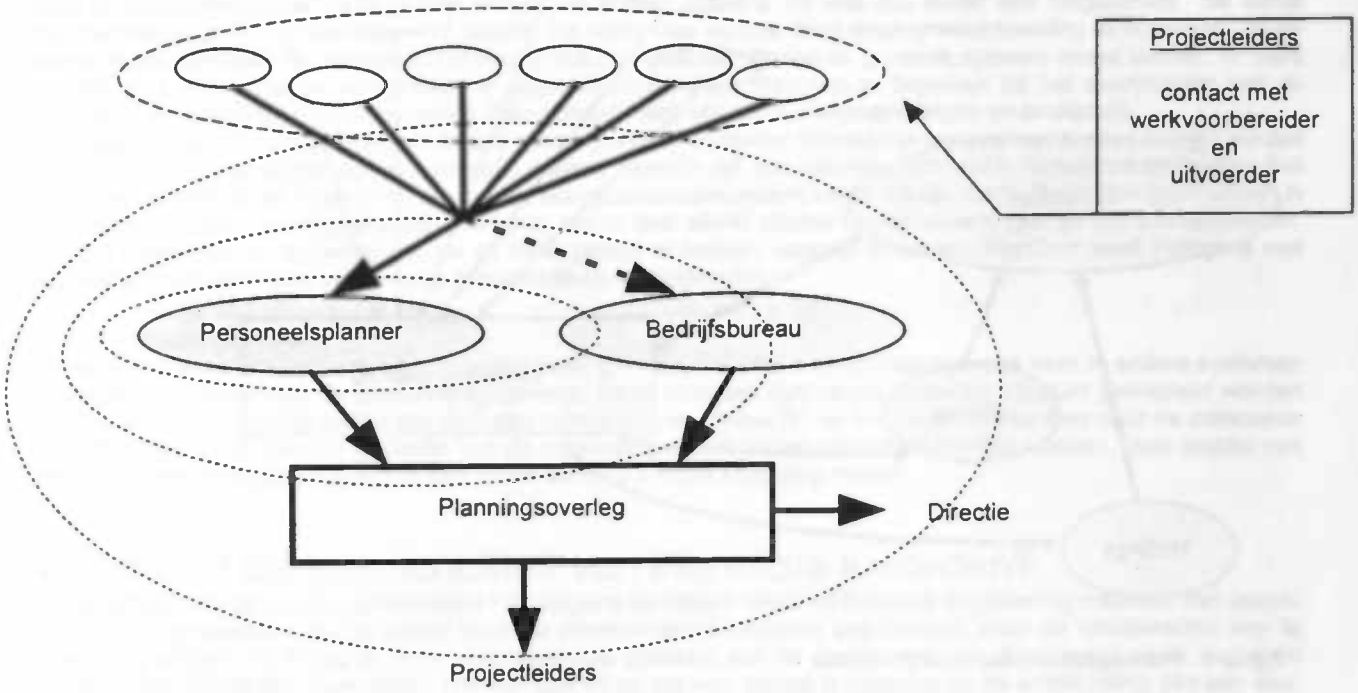
Verder hebben ze een kleine controlerende functie op de projectleiders. Deze hebben een aantal taken, die bij de grotere bedrijven door de personeelsplanner gedaan worden. We zien dat er bij Hendriks een soort overlapping van taken is, die plaatsvindt in onderling overleg. De belangrijkste taken zijn het toepassen van CAO regels, het egaliseren van pieken en dalen in de personeelsbehoefte en het in de gaten houden van ontslagtermijnen.
We konden stellen dat door de kleinere omvang van Hendriks in verhouding met Grootel en Moes, het onderling overleg belangrijker werd in het planningsproces. Dit had als nadeel voor PlanMate dat het roosteren moeilijker te ondersteunen zou zijn met een kennissysteem.

Het planningsproces bij van der Heijden
Bij Van der Heijden werd gepraat met Van der Heijden en Van Lier. Van der Heijden is binnenplanner, Van Lier is buitenplanner. Bouwbedrijf van der Heijden heeft ca. 60 medewerkers in vaste dienst en is daarmee in omvang kleiner dan de overige betrokken bouwbedrijven. Hiervan wordt geprofiteerd door het bouwbedrijf heel flexibel te laten zijn. Er kunnen en moeten voortdurend ad hoc beslissingen worden genomen en dat is denk ik ook de beste omschrijving voor het planningsproces.
Bij Van der Heijden was daarom op dat moment geen behoefte aan ondersteuning van de personeelsinzet met behulp van PlanMate, maar men had met het oog op de toekomst wel een zekere interesse.

Wat was de algemene planningssituatie?

Overeenkomsten in de planningsprocessen
Op het moment dat we de verschillende roosterprocessen naast elkaar legden, zagen we duidelijk overeenkomsten tussen bedrijven van eenzelfde omvang. Moes en Grootel hebben ongeveer dezelfde omvang (plusminus 300 timmerlieden) en de planningssituaties bij die bedrijven kwamen ook opmerkelijk overeen. Hendriks is al een kleiner bedrijf en had dan ook een planningssituatie die minder gestructureerd is. Wanneer we naar van der Heijden keken ziagen we dat de roostersituatie daar zo ad hoc is dat ondersteuning door middel van computersoftware de roostermaker zelfs meer tijd zou gaan kosten door het administratief bijhouden van gegevens die van belang zijn voor PlanMate II. De elementen die we bij Moes en Grootel vonden, komen wel terug bij de andere bedrijven, maar in een mindere mate.

PlanMate in de algemene planningssituatie
Figuur 4 geeft een algemeen beeld van de roostersituatie zoals die dus (volgens ons) bij bouwbedrijven voorkomt. We moeten hier wel opmerken dat we, wanneer we hadden willen vaststellen dat deze



Figuur 4 : Algemene planningssituatie

situatie bij alle bouwbedrijven zo is, we een grotere hoeveelheid bouwbedrijven hadden moeten nemen voor een analyse van de planningssituatie.
Wanneer we PlanMate willen gaan plaatsen, kunnen we het best naar de cirkels kijken die in de algemene plannningssituatie zijn aangebracht. De eerste versie van PlanMate II geeft binnen de binnenste cirkel de ondersteuning, en in de loop van de tijd zal deze cirkel steeds groter gaan worden, waardoor ook andere objecten in het planningsproces ondersteund zullen gaan worden.
De eerste versie van PlanMate II wordt dan ook wel de babyversie genoemd, wat treffend weergeeft dat het ontwikkelen van PlanMate gewoon door zal gaan zodat het uit kan groeien tot een volwassen roosterapplicatie.

Conclusie

Het is duidelijk dat er een vast patroon van plannen bestaat in de bouwwereld. Naast de lange termijn planning bestaat de korte termijn planning en de hele korte termijn planning. Van deze planningen zijn vooral de eerste twee goed te ondersteunen door een computer en is PlanMate II ontwikkeld om vooral te gebruiken wanneer voor de korte termijn een planning gemaakt moet worden. Bij deze planning is het in grote lijnen de bedoeling dat meerdere personen op meerdere bewerkingen ingeroosterd worden over een lange tijdsperiode.
PlanMate I is hierbij voornamelijk als inspiratiebron gebruikt doordat de bij de implementatie gebruikte ontwikkelomgeving niet overeenkwam met de ontwikkelomgeving die KPD gebruikt.
Het valt wel op dat we over middelgrote en grote bouwbedrijven praten wanneer we over die ondersteuning praten. Over het algemeen wordt het ondersteunen per computer van de planningen binnen een bouwbedrijf steeds minder zinvol wanneer de omvang van het bouwbedrijf in grootte afneemt.

Hoofdstuk drie

De behoeften bij de roostermakers

Wanneer we ondersteuning willen gaan geven aan de personeelsroostering in de bouw, is het uiteraard zeer nuttig te gaan praten met de personen die op dit moment verantwoordelijk zijn voor deze personeelsroosters. In de eerste plaats om te onderzoeken of er wel vraag bestaat.
In deze gesprekken met de verschillende roostermakers werd duidelijk dat er vraag is naar ondersteuning en kwam een bepaald behoeftebeeld naar voren dat uiteraard een grote invloed op PlanMate II zal hebben. Het is nodig deze behoeften serieus te nemen omdat deze groep roostermakers de gebruikers van het pakket zullen gaan worden. Deze wensen worden hieronder kort beschreven, en geprobeerd zal worden deze te vertalen naar de functionaliteit die PlanMate zal moeten bezitten.
Daarnaast had KPD natuurlijk z'n wensen op het gebied van koppelingen van PlanMate aan personeelsadministraties en andere planningspakketten en was het belangrijk voor Jan Glaudé en mij om vast te stellen wat naar onze mening de belangrijkste behoeften waren, zodat deze in de eerste versie geïmplementeerd zouden kunnen worden.

Wat vinden de bouwbedrijven

De behoeften bij Moes
Bij Moes had men een zeer duidelijk beeld over waar PlanMate II zou moeten gaan functioneren doordat men al betrokken was bij het project PlanMate I. De administratieve taken van Mevr. Schettler zouden moeten worden vereenvoudigd, zodat iemand anders ze eventueel zou kunnen overnemen, en er zouden één of meerdere voorstellen voor de personeelsplanning gegenereerd moeten kunnen worden die in de planningsvergadering ter ondersteuning zouden kunnen dienen. Daarom was ook duidelijke uitvoer naar de printer nodig. De plaats van PlanMate was dus duidelijk voor de planningsvergadering en moest een sterk administratief karakter hebben.

De behoeften bij Grootel
Bij Grootel had men wat moeite om de plaats van PlanMate te bepalen. Duidelijk was hier dat het administratieve gedeelte ook zeer belangrijk werd gevonden en dat voornamelijk de overzichtelijkheid van de gegevens en het gebruikersgemak een hoge prioriteit had. Een bouwplaatsmedewerker kan namelijk naast zijn discipline nog bepaalde cursussen en kwalificaties bezitten. Het bleek bij Grootel een belangrijke zaak dat PlanMate al deze gegevens kan verwerken en kan weergeven op het scherm en op papier. De betrokken personen bij Grootel hadden in eerste instantie weinig vertrouwen in ondersteuning op het gebied van personeelsplanning, maar zagen later ook enkele mogelijkheden tot beperkte ondersteuning bij het minimaliseren van de reiskosten.

De behoeften bij Hendriks
Bij Hendriks ging ook de meeste belangstelling uit naar een verbeterde administratie. De hele korte termijn wilde men niet ondersteund hebben door een computerprogramma, maar bij de middellange planning zag men er wel heil in. Gezien het planningsproces zal PlanMate bij Hendriks een meer controlerende functie krijgen doordat in het overleg tussen de projectleiders en de directie de personeelsplanning plaatsvindt en daarna de randvoorwaarden pas gecontroleerd worden. De roostermaker bij Hendriks had in tegenstelling tot die bij Grootel, meer oog voor de projectkant van de administratie en minder voor alle attributen die een werknemer kan hebben. Hier was de vraag naar verschillende methoden om bewerkingen van elkaar te onderscheiden op attribuutwaarden en werknemers en ploegen te volgen in de tijd belangrijker.

Conclusie
Er blijkt bij een aantal roostermakers een grote behoefte te zijn naar planningsondersteuning. Dit is uiteraard een gevolg van het feit dat deze tot nu toe op de markt ontbreekt.
De behoeften bij die bouwbedrijven zijn voornamelijk gericht op de administratieve ondersteuning. Ondersteuning door middel van een kennisbank zag men niet lukken, maar dat kwam grotendeels doordat men niet goed begreep hoe zo'n kennisbank werkt.
Het valt verder op dat door de afwijkende behoeften tussen Grootel (personeelsadministratie belangrijker) en Hendriks (projectadministratie belangrijker) de opties die in PlanMate geïmplementeerd zullen worden niet gericht zullen zijn op één gedeelte van de administratie, maar op alletwee.

PlanMate en andere planningspakketten

Andere planningspakketten
PlanMate moest gegevens uit andere planningssystemen, zoals PlanCalc, kunnen invoeren om zo de klant geen onnodig werk te bezorgen wanneer deze gebruik maakt van andere software. Dit was dus zuiver een mogelijkheid die vanuit commercieel oogpunt noodzakelijk werd geacht. De vraag was nu hoe deze gegevensoverheveling te realiseren. Bij de keuze uit alle denkbare antwoorden stond voorop dat PlanMate een flexibel en generiek systeem moest worden. Er werd daarom door Alfred Blaauw voorgesteld de gegevensstroom naar PlanMate te beperken tot het overbrengen van de activiteiten (= bewerkingen van een project). Elke activiteit heeft een reeds vastgesteld begin- en eindtijdstip.
Deze aanpak had als voordeel dat ook andere planningssystemen relatief gemakkelijk aangesloten zouden kunnen worden op PlanMate. Er was echter ook een mogelijk nadeel, namelijk dat een kennistechnologisch interessant onderwerp (het afvlakken van pieken en dalen in de vraag naar bouwplaatsmedewerkers) waarschijnlijk vooral plaats zou gaan vinden in het planningssysteem voorafgaand aan PlanMate, omdat daar allerlei relaties tussen bewerkingen gelegd kunnen worden. Het lijkt echter niet de bedoeling die, na ze eerst gelegd te hebben, weer te schenden. PlanCalc heeft overigens met betrekking tot dit onderwerp al de nodige administratieve mogelijkheden.

Personeelsadministraties
Bij het inroosteren van personeel op projectbewerkingen zijn niet alleen de projectgegevens vaak in andere systemen ondergebracht, maar ook de personeelsgegevens. Deze moesten dus vanuit hetzelfde oogpunt gekoppeld worden aan PlanMate. De praktijk leerde dat door een conversie van Dbase IV en Microsoft Access-files naar de databases van PlanMate al het grootste gedeelte van de personeelsadministraties gekoppeld konden worden. Door middel van een algemeen conversieprogramma kan zo elk bedrijf z'n eigen koppeling maken.

Wat moest de functionaliteit van PlanMate II worden?

In het behoeftemodel dat ik geproduceerd heb volgens de Object-Oriented Software Engineering methode van Jacobson[1] heb ik geprobeerd aan te geven waar de roostermakers behoefte aan hebben, door de functionaliteit aan te geven, die volgens mij nodig is. Ik heb hierbijgenruik gemaakt van de documenten die Jan Glaudé en ik opgesteld hebben [Glaudé/van der Meer 1994]. Het valt daarbij op dat een aanzet is gegeven tot de ontwikkeling van een evaluator, maar dat deze nog niet is uitgewerkt. Dit is intussen gedaan door Erik Huisman van het IKS [Huisman, 1994]. Ik zal in de volgende paragrafen de functionaliteit die onze rooster-editor zal moeten bezitten nog even kort vermelden en proberen aan te geven waarom hij nodig is.

Het gebruikersinterface
Het leek mij van belang het gebruikersinterface van PlanMate overeen te laten komen met dat van PlanCalc. Het weergeven van de invoer vanuit PlanCalc, op dezelfde manier als dat in PlanCalc gebeurt, zou voor de gebruiker een logische en overzichtelijke aansluiting tussen de twee pakketten geven. Ook de opbouw van het gebruikersinterface moest overeen gaan komen met PlanCalc en er is naar gestreefd de overzichtelijkheid optimaal te maken. Moeilijkheid hierbij was dat bij PlanCalc geen personeelsgegevens op het scherm nodig waren, terwijl een personeelsroosteraar projectgegevens én personeelsgegevens nodig heeft. Dit maakte de opbouw van het gebruikersinterface al een stuk moeilijker.

Administratieve taken
Een roostermaker heeft dus minstens drie bestanden nodig:
- Een personeelsbestand
- Een projectenbestand
- Een roosterbestand

Er moest daarom de mogelijkheid bestaan een personeelsbestand aan te leggen, waarbij alle werknemers een aantal attributen hebben. Er zijn een aantal vaste attributen, die bij het genereren van personeelsroosters een rol spelen, maar er moeten ook attributen ingevoerd kunnen worden, en die verschillen dus per bedrijf. Deze laatste groep is vanzelfsprekend niet van invloed op het kennissysteem.
Het moest ook mogelijk zijn een projectenbestand aan te leggen, maar we zagen het toevoegen en verwijderen van fasen in het bouwproces niet als een administratieve taak van PlanMate. Dit gebeurt namelijk in PlanCalc al. PlanMate kan wel signaleren dat door een piek of dal in de personeelsvraag er geschoven moet worden met subfasen, maar dan moet door de roosteraar teruggeschakeld worden om dit dan daadwerkelijk, met de hand, te doen. Wilden we dit door PlanMate laten doen, dan werd het al gauw een heel complex geheel en was het zelfs maar de vraag of dit mogelijk was, daar de indeling in bouwfasen per bouwbedrijf redelijk verschilt en dus moeilijk in een generiek systeem onder te brengen is. Door het toekennen van personeel aan bewerkingen creëert de roostermaker uiteindelijk het roosterbestand.

[1] Jacobson ontwikkelde een objectgeörienteerde systeemontwikkelmethode waarmee door middel van analyse, ontwerp en implementatie een overzichtelijk beeld gegeven wordt van de ontwikkeling van een softwarepakket.

Het moest ook mogelijk zijn dit in te laden uit andere software, maar daar kom ik bij het ontwikkelen van de generator op terug. Bij het bepalen van de attributen, die volgens Jan Glaudé en mij van belang zijn voor de evaluator, controleur en generator is uiteraard gekeken naar de attributen die op dit moment gebruikt worden bij de bouwbedrijven. Welke deze attributen zijn volgt in de paragraaf over de uiteindelijke structuur van de databases.

Tellingen
Een aantal tellingen zullen belangrijk zijn voor het te realiseren kennissysteem en het spreekt voor zich dat deze geïmplementeerd zijn. Als belangrijkste tellingen spreken we dan van: Het tellen van de personeelsbehoefte. Het berekenen van reistijden en andere personeelsgebonden variabelen (te denken valt aan spreadsheetachtige functies). Dit is voornamelijk belangrijk voor de controle op bepaalde Cao-regels. Dit is het onderdeel van de functionaliteit, die op een goede basis van PlanMate II steeds meer uitgebreid zal gaan worden. De bovenstaande tellingen zijn in eerste instantie de belangrijksten, maar in de loop van het project zijn daar al weer nieuwe bijgekomen. Deze zijn echter al gauw te rekenen tot het berekenen van personeelsgebonden variabelen omdat dat een veelomvattend begrip is.

Conclusie
Met het implementeren van de bovengenoemde punten in PlanMate II was het volgens mij mogelijk een redelijk generiek systeem te realiseren dat een duidelijke toegevoegde waarde voor PlanMate I had. Het systeem deed in de prototypeversie nog karig aan, daar het genererende gedeelte nog ontbrak en bepaalde (minder relevante) tellingen en administratieve mogelijkheden nog geïmplementeerd moesten worden Wanneer er echter een goede basis ligt, is deze functionaliteit makkelijk toe te voegen aan PlanMate II.

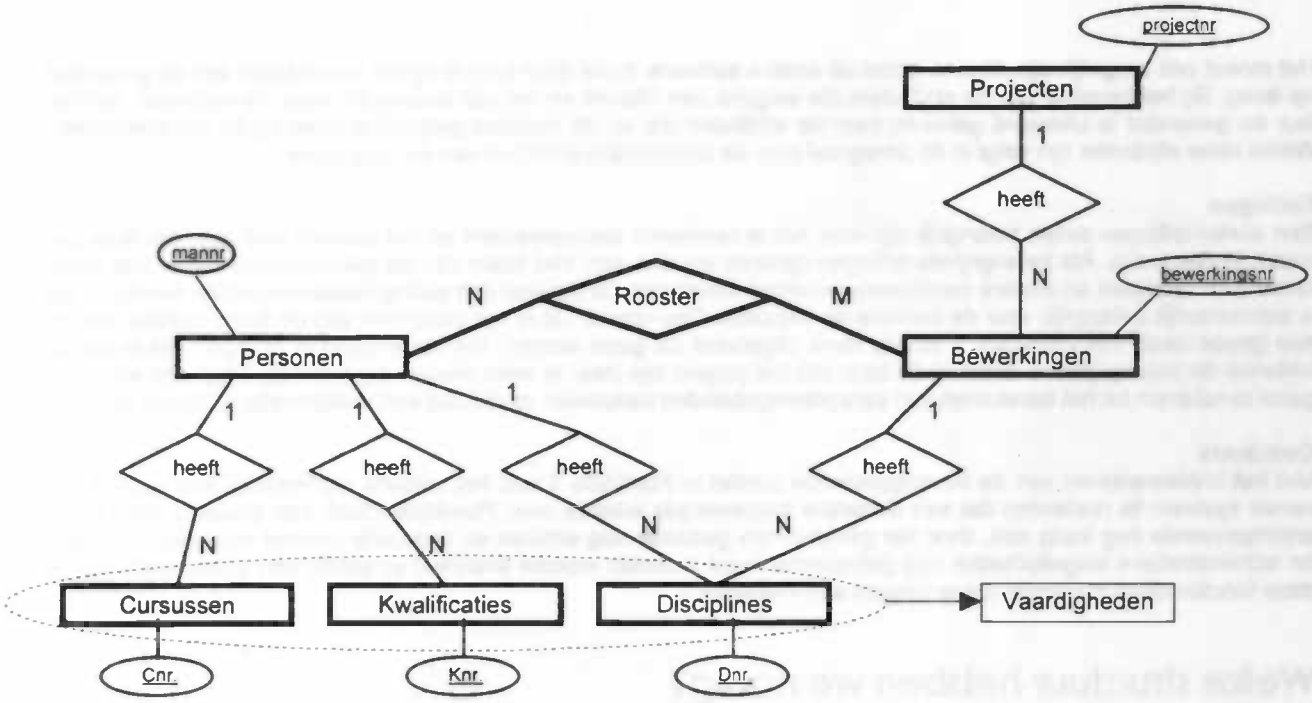
Welke structuur hebben we nodig?

Eigenlijk is PlanMate II als rooster-editor niets meer dan een mooie bovenkant op een database, die de functionaliteit bezit om te bepalen welke gegevens getoond moeten worden, hoe ze eruit moeten zien en of ze aan elkaar gekoppeld moeten worden. Met de eerdergenoemde methode van Jacobson heb ik het hoe en waarom van de structuur van PlanMate en de onderliggende databases uiteengezet. Ik zal kort de belangrijkste zaken uit die modellering vermelden.

De database onder PlanMate II
Het aanmaken van een goede database was belangrijk, omdat het moeilijk en kostbaar zou zijn deze in de toekomst te veranderen. Probleem hierbij was dat de verschillende bouwbedrijven verschillende hoeveelheden attributen per werknemer gebruiken. Om dan een generiek systeem te ontwikkelen is het nodig zelf attributen te kunnen toevoegen en verwijderen. Dankzij de mogelijkheden in Visual Basic (zie deel 3) is dit mogelijk, maar we wilden natuurlijk wel een database hebben die in een hogere normaalvorm zou zijn dan de eerste. Van de attributen die toegevoegd zullen gaan worden, zijn dan vooral de vaardigheden (disciplines, cursussen en kwalificaties) velden die veel leeg zullen zijn, of juist meerdere waarden moeten bevatten. Dit komt de overzichtelijkheid niet ten goede. Ik heb er dan ook voor gekozen deze velden in tabellen te zetten. Andere velden worden toegevoegd aan de tabel 'Personen' en daar kan dus een overschot aan Null-waarden optreden. Dit is moeilijk tegen te gaan zonder ook de mogelijkheid te bieden nieuwe tabellen aan te maken. Deze optie is nu nog altijd mogelijk, maar is voorlopig niet van belang en te bewerkelijk om even toe te voegen.

Een model van de databases
Wanneer we de beginsituatie met een entity-relationship model, met slechts de key attributen, weergeven ziet dat eruit als in Figuur 5.
Een overzicht van de belangrijkste tabellen met hun velden is opgenomen in bijlage A.
Door deze opzet is het mogelijk veel vaardigheden in te voeren en toe te kennen aan personen, zonder dat veel lege velden ontstaan.

Welke uitbreidingen kunnen we verwachten in de database?
Het bleek handig gevonden te worden om instellingen in het gebruikersinterface te bewaren. Om deze instellingen te behouden moesten we dus ook een aantal tabellen maken. Deze zijn echter niet zo van belang en daarom vermeld ik ze hier niet.
Wanneer we (met of zonder een afstandstabel) de reisafstanden wilden opslaan, zodat we deze de volgende keer niet opnieuw hoeven in te voeren, was ook een database nodig. Hetzelfde geldt voor CAO-tabellen die gebruikt zullen gaan worden bij de evaluator, controleur en generator. Deze tabellen spreken echter voor zich en hebben verder geen relaties met de bovenstaande tabellen van PlanMate II. Ze worden alleen gebruikt om informatie uit te halen en eventueel in te schrijven.

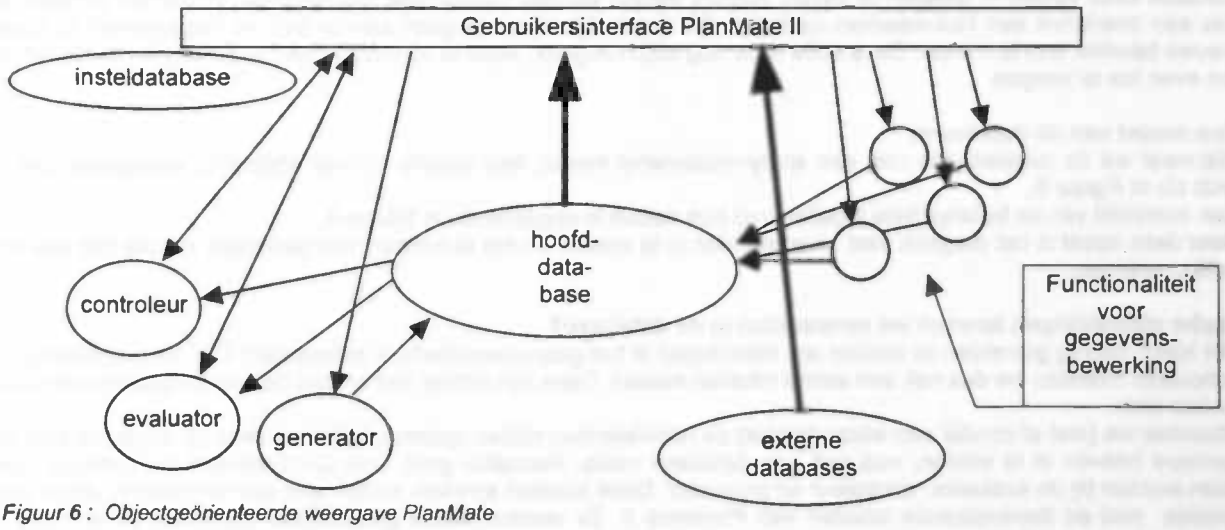


Figuur 5 : Beknopt schema databases PlanMate

In de oude versie van PlanMate was de mogelijkheid opgenomen om per bewerking en per persoon slechts één tijdsperiode mee te geven. Dit is intussen zodanig aangepast, dat in het rooster en bij de bewerkingen meerdere tijdsperiodes gedefinieerd kunnen worden. Deze zogenaamde discontinuïteit had uiteraard ook gevolgen voor de controleur en evaluator.

De gehele structuur van PlanMate II

Bovenop de database kwam dus een mooie schil waardoor er goed met de gegevens uit de databases gewerkt kon worden. Dit kunnen we, net als de database, als losstaand object beschouwen, wat alleen maar door de gebruiker is te bedienen, maar waar door andere objecten door middel van de database, gegevens ingevoerd kunnen worden. Dit is van belang bij een eventuele generator. Deze zou door het gegenereerde rooster in PlanMate-formaat op te slaan, aan PlanMate gekoppeld kunnen worden.



Figuur 6 : Objectgeörienteerde weergave PlanMate

We hebben nu dus een sterk objectgeörienteerd stuk software. Hier werd voor gekozen, zodat uitbreidingen aan het produkt te koppelen zijn, en geen aanpassingen in de software vergen. De functionaliteit van het gebruikersinterface reken ik ook tot deze objecten, waarbij soms wel een kleine aanpassing nodig is in de andere software. Deze aanpak is belangrijk met het oog op de toekomst. Over het algemeen is de levensduur van een software-pakket namelijk niet erg lang. Veranderingen in de omgeving van de software (onder andere door de software zelf) stellen meer en/of

andere eisen aan het pakket waardoor vaak een nieuwe functionaliteit nodig is. Als dit elke keer betekent dat het gehele programma aangepast dient te worden, is dit niet rendabel [Jacobson, 1993]. Deze aanpak biedt de mogelijkheid makkelijker zaken te veranderen en toe te voegen, waardoor de levensduur van de software verlengd kan worden.

Een objectgeörienteerd model van PlanMate
In een plaatje ziet Planmate er vervolgens zo uit als fFiguur 6. De modellering van PlanMate II wordt uitgebreid behandeld in het analysemodel [van der Meer 1994] dat volgde na het behoeftemodel. Hierin worden de belangrijkste onderdelen van de functionaliteit aangegeven zoals die in het eerste prototype van PlanMate gereed waren.

Conclusie
Voor het modelleren van PlanMate is voor een objectgeörienteerde aanpak gekozen. Niet omdat iedereen dat tegenwoordig lijkt te doen, maar omdat de aanpak voordelen heeft bij de verdere ontwikkeling van de software. Op deze manier kan bijvoorbeeld het gebruikersinterface makkelijk vervangen worden wanneer er in de toekomst niet meer mee te werken blijkt te zijn.

Deel 2

De kennisacquisitie,
kennisstructurering
en
Nexpert

Inleiding

Eén van de belangrijkste doelen van PlanMate II was het implementeren van een generator die door middel van kennisbanken in staat moest zijn een rooster te ontwikkelen wat de roostermaker kan ondersteunen.
Wanneer we een kennissysteem willen ontwikkelen is het kortgezegd de bedoeling de computer dezelfde procedures en stappen te laten nemen als de expert en hem op die manier beperkt te laten redeneren. In hoofdstuk vier zal ik wat achtergrondinformatie proberen te geven over kennissystemen in het algemeen.
Voordat we over kunnen gaan op de implementatie van de kennis van de roosterexperts in de bouwwereld, zijn er een aantal interviews nodig waarin in de eerste plaats een inventarisatie plaatsvindt van de inhoud en omvang van het kennisgebied van de expert en de problemen die hij of zij daar tegenkomt.
Ik zal in hoofdstuk vijf dieper ingaan op de acquisitie en de daaruit verkregen regels en behoeften die volgens Jan Glaudé en mij van belang waren bij de ontwikkeling van PlanMate II. Het was bij deze fase van ongestructureerde en gestructureerde interviews vooral de taak van Jan Glaudé om de kennis uit de roostermakers te krijgen, dus zal ik veel van zijn gegevens gebruiken in dit gedeelte van het verslag.
In hoofdstuk zes, waar de structurering van deze kennis beschreven wordt, komt de informatie van Jan Glaudé weer naar voren, hoewel ik het omzetten van de informele regels naar Nexpert-invoer zelf heb gedaan.
In hoofdstuk zes zal ik ook dieper op de ontwikkelshell Nexpert ingaan, omdat volgens de structuur van deze shell de ontwikkeling van de kennisbanken zal gaan plaatsvinden en de structurering daar dus op afgestemd moet worden. Duidelijk wordt waarom voor Nexpert is gekozen, maar ook waarom Nexpert niet goed te koppelen is aan een commercieel produkt. Wanneer we weten hoe de invoer van de ontwikkelshell er uit moet zien, kunnen we de kennis gaan structureren.
Het implementeren van de controleur en evaluator, kan gebeuren door middel van routines in gewone programmacode. Een korte beschrijving van deze routines is opgenomen in hoofdstuk zeven.

Hoofdstuk vier

Kennissystemen

Wanneer we willen gaan bestuderen of de personeelsplanning binnen de bouw te ondersteunen is door middel van een (beperkt) kennissysteem, moeten we eerst vaststellen wat we precies onder een kennissysteem verstaan.

Definities van kennissystemen

Zijn er wel definities?
Hoewel de term kennissystemen vaak gebruikt wordt, is er eigenlijk geen algemeen aanvaarde definitie van kennissystemen. Dit hoeft op zich niet zo'n probleem te zijn daar in de praktijk blijkt dat het praktisch gebruik van deze term niet gehinderd wordt door het ontbreken van die definitie. Johan Oldenkamp [1993] heeft een studie gedaan naar definities van kennissystemen en hij zet daarin 28 definities naast elkaar die door deskundigen op dit gebied tot nu toe zijn geformuleerd. Het heeft geen zin dat hier allemaal te herhalen, dus ik zal zijn conclusie overnemen en ik zal proberen mijn eigen definitie van een kennissysteem te formuleren.

Een algemene definitie
Oldenkamp herkent twee aspecten in de verzameling van definities. Ten eerste het functionele aspect dat beschrijft wat een kennissysteem volgens de deskundigen moet kunnen, en daarnaast het architectonische aspect van kennissystemen waarin we de opbouw terugvinden.
In de algemene **definitie** zitten uiteraard beide aspecten verwerkt en hij luidt als volgt:

> *Een kennissysteem is een computerprogramma dat op grond van informatie autonoom of door interactie met een eindgebruiker, problemen op kan lossen en hierover uitleg kan geven door met behulp van een redeneermechanisme gebruik te maken van expliciet en gescheiden gerepresenteerde kennis.*
>
> Oldenkamp 1993

Persoonlijk zie ik de koppeling van een kennissysteem aan een computerprogramma niet zo strak en in de discussie komt Oldenkamp hier ook nog even op terug. Duidelijk is dat we hier terecht komen op het gebied van systeemleer en de discussie wat we onder een systeem verstaan. Ik zie dus meerdere mogelijkheden voor systemen zoals bijvoorbeeld een combinatie van computer en personen.

Een eigen definitie
In mijn definitie zal ik dus niet eisen dat een kennissysteem een computerprogramma moet zijn. Zo zie ik bijvoorbeeld ook een blindegeleide hond als een beperkt systeem dat met kennis van ziende mensen, verkeersproblemen kan oplossen voor z'n gebruiker door procedures toe te passen die ingevoerd zijn door training. Het redeneermechanisme van de hond is uiteraard beperkt tot één stap van signaleren, classificeren en reageren, maar in feite is dat het toepassen van kennis van een domeindeskundige. Hierbij moet opgemerkt worden dat de hond niet bewust zal classificeren met wat voor een probleem hij of zij te maken heeft.
Hierin ligt ook de kern van mijn eigen definitie: het toepassen van regels (hoe beperkt en klein ook) als reactie op een bepaalde situatie, waarbij bij de ontwikkeling van het systeem de koppeling is gelegd tussen de situatie en de bijbehorende reactie. De verschillende redeneermechanismen bepalen in welke volgorde deze handelingen uitgevoerd moeten worden en hoe gereageerd moet worden op veranderingen die, door die reactie, in de omgeving optreden.
Laten we de verschillende onderdelen even definiëren:

- **Een stap**: een logische operatie, een toekenning aan een variabele of een aanroep van een standaardprocedure.
- **Een regel**: een logische waarde die aangeeft of de validatie van een groep van stappen wel of niet gelukt is.
- **Een redeneermechanisme**: het bepalen van de volgorde waarin regels en stappen gevalideerd moeten worden en hoe gereageerd moet worden op de veranderingen die deze validaties teweeg brengen.

Met deze aannames is volgens mij een **definitie** van een kennissysteem:

> *Een systeem waarin, binnen een bepaald domein, de handelwijze van één of meer domeindeskundigen is uitgesplitst in stappen en regels, die zodanig gegroepeerd zijn dat ze door een redeneermechanisme in een bepaalde volgorde worden gevalideerd met als doel één of meer oplossingen te genereren.*

Dit is duidelijk een meer praktijkgerichte definitie, maar bevat volgens mij wel de kern van een kennissysteem.

Verschillen in opvatting
Het valt op dat ik niet eis dat een kennissysteem kan uitleggen waarom het iets doet. Dit komt volgens mij namelijk neer op het bijhouden van de gevalideerde stappen. De ontwikkelaar van het systeem geeft op dat moment door middel van een stukje tekst aan waarom hij of zij het gesignaleerde probleem aan de betreffende regel heeft gekoppeld en de ontwikkelaar van het redeneermechanisme geeft aan waarom er op een bepaalde manier gevalideerd wordt. Meer implementeren in het systeem is niet mogelijk en het heeft dus in feite niets te maken met het systeem op zich, maar met een stukje gebruikersvriendelijke informatie naar de gebruiker.
We zouden bijvoorbeeld de hond kunnen leren dat hij of zij moet blaffen wanneer hij of zij stopt voor een weg. Wij weten dan dat het beest daarmee bedoelt dat er gestopt wordt omdat er verkeer over deze weg kan rijden, dat door de hoge snelheid een gevaar kan zijn wanneer we zomaar zouden oversteken en wij daarom onze blindestok omhoog moeten steken, zodat het langsrazende verkeer voor ons moet stoppen. Was ons echter niet vertelt wat één keer blaffen betekent en we merken dat de hond stopt en blaft omdat hij of zij waarschijnlijk een stoeprand heeft gesignaleerd, dan lopen we (neem ik aan) niet opeens door. Het implementeren van een uitlegfaciliteit is daarom volgens mij een apart kennissysteem dat overlapt met het bepaalde kennisdomein doordat het kennis bevat over het kennissysteem.
Ook de mogelijkheid tot interactie met de gebruiker is niet een eigenschap van een kennissysteem die geëist dient te worden. Het is zelfs een beperking van een kennissysteem wanneer het niet op eigen kracht een oplossing zal weten te genereren, maar gestuurd moet worden door een domeindeskundige. Uiteraard zal interactie met de gebruiker altijd nodig blijven, maar het blijft één van de manieren om informatie te vergaren, die nodig is om een stap en/of regel te valideren.

Kennissystemen op planningsgebied

PlanMate moet ook ondersteuning gaan bieden door middel van een kennissysteem. Het is dus interessant te gaan kijken in hoeverre er al ervaring met andere kennissystemen is, die van nut kan zijn bij de ontwikkeling van de kennisbanken van PlanMate.

DISKUS
Omdat roosterplanning één van de meest complexe cognitieve taken is die door mensen wordt uitgevoerd, is er op dit gebied veel behoefte aan ondersteuning voor roostermakers. De stichting DISKUS (Dynamic Interactive Scheduling and Knowledge Utilization Systems) houdt zich bezig met onderzoek op dit gebied. Dit kan enerzijds al grotendeels door met behulp van een goede en inzichtelijke representatie in het gebruikersinterface het probleem overzichtelijker te maken.
Anderzijds is het mogelijk het geproduceerde rooster te controleren op een aantal belangrijke randvoorwaarden en het te evalueren op de mate waarin bepaalde kwaliteitsaspecten zijn gerealiseerd. Tenslotte zou met een genererende component een aantal voorstellen gegenereerd kunnen worden, die als mogelijke invulling meegenomen kunnen worden in het planningsproces.

Het SEC-model
Om meer inzicht te krijgen op planningsproblematiek is er binnen DISKUS het Schedulings Expertise Centrum model ontwikkeld. Dit zogenaamde SEC-model is in feite een databank waarin volgens een bepaalde classificatie oplossingen voor planningssystemen onder kunnen worden gebracht. Dit zal vooral met het oog op het hergebruik van software een belangrijke bron moeten worden waaruit men in de toekomst kan putten.
Ik zal alleen de belangrijkste zaken uit dit model verduidelijken, omdat het classificeren van het planningsprobleem een taak is voor de beheerder van het SEC-model en (nog) niet relevant is bij PlanMate II:

- **Schedulingsrandvoorwaarde (SRV).** Deze regel mag de roostermaker niet schenden. Hier bestaat een onderscheid tussen impliciete (onmogelijkheden) en expliciete regels (afgesproken zaken).
- **Schedulingsdoelfunctie (SDF).** Bij deze regel wordt gestreefd naar een optimale realisatie, maar deze mag dus wel geschonden worden.

Zoals al eerder genoemd, spelen deze regels een belangrijke rol bij het controleren, evalueren en genereren van planningen. We zullen deze begrippen dan ook nog vaker tegenkomen.

Roosterapplicaties volgens het SEC-model
Wanneer we een kennissysteem ter ondersteuning van een planningsapplicatie willen ontwikkelen, kunnen we een aantal typen applicaties onderscheiden [Jorna/Weezel/Huisman, 1994]. Ik zal ze kort behandelen naar graad van 'intelligentie':

1. **Rooster-editor:** Kort vertaalt spreken we hier van een elektronisch planbord voor de roostermaker. Het is dus een gebruikersinterface met daaronder de benodigde databases waarop een aantal bewerkingen en berekeningen uitgevoerd kunnen worden. Zoals we al eerder aanstipten kan hiermee de roostertaak in belangrijke mate ondersteund worden doordat de overzichtelijkheid vergroot wordt.
2. **Rooster-controleur:** Dit is een rooster-editor met een toegevoegde functie die een controlerende werking heeft. Er wordt gecontroleerd of randvoorwaarden niet geschonden worden.

3. **Rooster-evaluator:** Hier is een functie toegevoegd, die beoordeelt in hoeverre de doelfuncties gerealiseerd zijn. Wanneer de bovengenoemde functies ook geïmplementeerd zijn, spreken we van **een rooster-auditor**.
4. **Rooster-interactor:** Een roosterapplicatie die de status van rooster-interactor heeft kan zelf roosters genereren. Doordat de roostermaker deze roosters nog zal moeten kunnen aanpassen blijven we spreken van een interactor.

Met deze opsomming geven we eigenlijk ook het traject weer dat tot nu toe bij het ontwikkelen van kennissystemen vrijwel altijd gevolgd wordt. Dit komt mede doordat de graad van complexiteit meer toeneemt dan de graad van 'intelligentie'.

Conclusie

Op de vraag wat een kennissysteem nu precies is, is niet goed een algemene definitie te geven. Dit is in de praktijk echter geen echte belemmering gebleken bij de ontwikkeling van kennissystemen.
Vooral in roosterapplicaties blijkt het kennissysteem een geschikte ondersteuning te kunnen bieden en de stichting DISKUS houdt zich er op wetenschappelijk gebied mee bezig hoe dit zo efficiënt en goed mogelijk gerealiseerd kan worden. Vandaar dat ook het project PlanMate II een planningsprobleem behelst.

Hoofdstuk vijf

De kennisacquisitie

De fase van kennisacquisitie is de fase waarin we moeten zien uit te vinden hoe roostermakers precies te werk gaan bij het vormen van een personeelsplan. Deze kennis is niet alleen belangrijk bij het ontwikkelen van de kennisbanken, maar is ook erg nuttig bij de ontwikkeling van het gebruikersinterface.

Kennisacquisitie in het algemeen

Wat is kennisacquisitie?
Onder kennisacquisitie verstaan we kortweg het onderzoeken welk kennisdomein de basis vormt van de activiteiten van de roostermaker en hoe hij deze kennis toepast. Het is hier belangrijk de in het kennisdomein belangrijke entiteiten en bewerkingen vast te stellen. Dit is voor mij één van de interessantste fasen in de software engineering, omdat het hier nodig is om de roostermaker op zijn vakgebied te kunnen begrijpen en dat te vertalen naar de ontwikkelomgeving van het softwarepakket. We zien deze fase in feite bij elke systeemontwikkelmethode terug als inleidende fase om de kennis van de expert in kaart te kunnen brengen en er zijn in de literatuur een aantal elicitatietechnieken te vinden die dit op verschillende manieren proberen te bereiken.

Methoden van kennisacquisitie
In de literatuur zijn een grote hoeveelheid kennisacquisitietechnieken terug te vinden. Het gaat te ver deze allen te behandelen (belangstellenden verwijs ik naar Jorna/Simons hfd. 5 en 6), maar de grote lijnen zal ik voor de duidelijkheid even vermelden:

- directe niet structurerende technieken: We spreken hier van ongestructureerde interviews of van protocolanalyse om het kennisdomein van de domeindeskundige te inventariseren. Bij de laatste probeert de deskundige hardop te denken bij het uitvoeren van zijn taak en onthult daar voor de systeemontwikkelaar al hoe deze het probleemdomein zou moeten opsplitsen in stappen en regels.
- directe structurerende technieken: Deze technieken gieten de uitkomst van de kennisacquisitie in een bepaalde vorm. Hier zijn het gestructureerde interview (vragenlijsten etc.) en doelontleding (de expert moet door vragen achter het probleem zien te komen) voorbeelden van.
- indirecte structurerende technieken: Methoden om de niet 'zichtbare' structuur in de kennis bloot te leggen. Het gaat hier voornamelijk om schaaltechnieken en het bepalen van prioriteiten binnen het kennisdomein.

Het is duidelijk dat het een zekere vaardigheid vereist om goed kennis te kunnen eliciteren, en dat via het toepassen van de juiste technieken veel belangrijke informatie boven tafel gekregen kan worden.

Waar moet je op letten bij het plegen van kennisacquisitie?
Het is bij het plegen van kenniselicitatie belangrijk een aantal dingen in de gaten te houden. Bij een interview komen namelijk een aantal psychologische aspecten kijken, die de kennisacquisitie kunnen beïnvloeden.
Ten eerste kan de interviewer zijn vragen zodanig stellen dat de geïnterviewde een bepaalde richting op gestuurd wordt.
Daarnaast is het voor veel experts moeilijk om precies onder woorden te brengen wat ze eigenlijk doen en wanneer ze er dan dieper over gaan nadenken is het heel goed mogelijk dat ze de procedures, die ze volgen, vervormen.
Uiteindelijk is de manier waarop de interviewer het antwoord verwerkt of begrijpt ook niet vrij van fouten en kunnen er daardoor onzuiverheden in de kennisbanken optreden.

De kennisacquisitie bij PlanMate II

De gebruikte methoden
De kennisacquisitie bij het project PlanMate II werd uitgevoerd door Jan Glaudé die als stagiare van IKS meer ervaring had bij het plegen van kennisacquisitie dan ik. Ik ben echter vaak meegegaan om te kijken hoe deze acquisitie in zijn werk ging om zo een beeld van de omgeving te krijgen waarin ik aan het werk moest.
Jan Glaudé gebruikte directe niet structurerende technieken bij de kennisacquisitie door de roostermakers te interviewen. Het bleek moeilijk een hardop-denkprotocolanalyse uit te voeren omdat de roostermakers eigenlijk vooral de administratieve kant van het planningsoverleg bleken te zijn, en dus zelf geen roosterstappen namen. Als structurerende techniek kan ik de vragenlijst noemen die we rondgestuurd hebben. Hierop konden de betrokkenen aangeven welke functionaliteit ze in PlanMate wensten.
Ik vind het achteraf gezien jammer dat we geen andere technieken hebben toegepast, want ik ben er van overtuigd dat de roostermakers best in staat waren geweest zelf een rooster te produceren dat ook uit het planningsoverleg zou

zijn gekomen. Dit had misschien kunnen onthullen hoe bijvoorbeeld overgebleven aanbod aan vraag toegewezen wordt. Op dit moment is hier nog niets van bekend en dat is jammer omdat we nu ook niet goed weten of er nu voorwaarts of achterwaarts geredeneerd wordt. Er zijn wel duidelijke aanwijzingen dat er voornamelijk voorwaarts geredeneerd wordt. Men heeft een vraag op een werk en gaat door met aanbod zoeken totdat deze vraag voldaan is. Dit is echter in de toekomst een punt waar verder onderzoek naar kan worden verricht.

De uitkomst van de acquisitie
Jan Glaudé en ik hebben van elk interview een verslag geschreven. Hieruit heeft Jan Glaudé een aantal regels gefilterd die volgens hem van belang zijn bij het controleren, evalueren en genereren van het rooster. Deze heb ik in bijlage B opgenomen. Naast deze regels is natuurlijk ook de hele ontwerp- en implementatiefase van het gebruikersinterface en de databases uit deze gesprekken voortgekomen.
Uit deze verslagen komen een aantal punten naar voren die bij het ontwikkelen van de controleur, evaluator en generator een rol zullen spelen.

De controleur
Het gaat hier dus om regels waarvan aangenomen mag worden dat ze niet geschonden mogen worden (de randvoorwaarden):

- **Impliciete regels**: Naast de databasetechnische regels (zoals het niet kunnen invoeren van een dubbel keyattribuut) moet er ook gecontroleerd worden op de volgende regels:
    - een persoon mag niet op twee plaatsen tegelijk ingeroosterd worden
    - bewerkingen moeten binnen de tijdsperiode van het gehele project vallen
    - iemand die ingeroosterd is, is niet vrij voor andere bewerkingen
    - het eind van een periode ligt na het begin van diezelfde periode
- **CAO en andere regels:** De belangrijkste door de roostermakers genoemde zaken van belang zijn hieronder genoemd. Andere zaken blijken vaak zoveel gedetailleerde gegevens van een werknemer te eisen, dat ze vrijwel niet te implementeren zijn, mits de roostermaker deze gegevens allemaal gaat bijhouden:
    - ontslagtermijnen
    - er moet altijd een EHBO'er en een voorman op een werkplaats aanwezig zijn
    - reiskosten zijn een grote uitgavenpost en moeten dus geminimaliseerd worden
- **Voldoen aan personeelsvraag:** De in de eerste stappen van het planningsproces opgestelde behoeftevraag dient in het rooster precies te worden ingevuld. Mocht blijken dat een extra mannetje nodig is, dan kan dit wel, maar een waarschuwing is het gevolg.

De evaluator
De doelfuncties die een roostermaker zo optimaal mogelijk probeert te realiseren zijn de volgende:

- **Samenstellen van de juiste ploeg:** Bij het samenstellen van ploegen is het op het eerste gezicht van belang te kijken of de juiste vaardigheden op de juiste bewerking terechtkomen. Daarna spelen de reisafstanden tussen het werk en de werknemers de grootste rol.
- **Minimaliseren van de reiskosten:** Bij het samenstellen van een ploeg is het voor iedereen voordelig wanneer de werknemers zo weinig mogelijk moeten reizen. Zeker gezien de hoge kostenpost die dit onderdeel altijd is geweest is het voor de bouwbedrijven belangrijk dat deze geminimaliseerd worden.
- **Optimaliseren bezetting auto of bus:** 'Carpoolen' werkt uiteraard kostenbesparend en niet alleen de afstand van werknemers naar het werk speelt dus een rol, maar ook de afstand van werknemers ten opzichte van elkaar, zodat ze met elkaar mee kunnen rijden.
- **Afvlakken van pieken en dalen in de personeelsbehoefte:** Wanneer door een dal in de personeelsbehoefte een aantal werknemers niet aan de slag kunnen, levert dit een verliespost op. Het is dus zaak dit te voorkomen. Evengoed is een vraag naar werknemers duur omdat er dan externe krachten ingehuurd moeten worden. We kunnen deze problemen oplossen door ze tijdig te signaleren en aan te geven welke personen ontslag aangezegd moet worden, of waar geschoven zal moeten worden om een dal met een piek te vullen.
- **Optimale continuïteit:** Het aan het werk houden van de werknemers is uiteraard het voordeligst voor elk bedrijf, maar ook een ploeg op hetzelfde werk proberen te houden scheelt inwerkkosten. Continuïteit is dus ruwweg iedereen zoveel mogelijk laten werken en zo weinig mogelijk van werkplek laten veranderen.

De generator
Hier spelen de randvoorwaarden en doelfuncties een rol zoals die bij de controleur en evaluator genoemd werden. Vooral met de randvoorwaarden moet een uitgebreide selectie plaatsvinden waardoor onze probleemruimte ingeperkt wordt. Met de doelfuncties kunnen we deze ruimte verder verkleinen, waarna een optimaal voorstel uit het beschikbare bestand gegenereerd kan worden. Lukt dit niet, dan vergroten we onze probleemruimte door bepaalde doelfuncties iets minder te waarderen. Hiervoor is het van belang dat we de waarde per doelfunctie kunnen instellen binnen PlanMate II.

De produktieregels voor PlanMate II
De regels die Jan Glaudé heeft geproduceerd zijn produktieregels (ALS ... DAN regels) die uit twee onderdelen bestaan. Het ALS-gedeelte (de condities) waarin een aantal condities opgenomen zijn, en het DAN-gedeelte (de acties) wat uitgevoerd wordt zodra de condities allemaal de waarde **true** leveren. Een voorbeeld:

ALS het bouwplaatspersoneel beschikbaar is
EN het bouwplaatspersoneel beschikt gegeven de behoefte over de juiste vakbekwaamheid
DAN kan het bouwplaatspersoneel gekozen worden voor inroostering op de bewerking

Naast het feit dat er een redeneermechanisme ontbreekt, zou deze regel niet in te voeren zijn in een hogere programmeertaal. Daarom is het dus nodig te gaan kijken wat een ontwikkelshell voor kennissystemen ons kan bieden en hoe de invoer er daar uit zou moeten zien. In het volgende hoofdstuk ga ik daar dieper op in.

Conclusie

Bij de kennisacquisitie die binnen het project PlanMate II heeft plaatsgevonden, zijn een aantal zaken naar voren gekomen die een grote rol spelen. Daarbij bleken de interne en externe bedrijfskundige kriteria van de planning de belangrijkste rol te spelen omdat hiermee kosten bespaard moesten kunnen worden. Bij PlanMate bleken vooral de reiskosten en de continuiteit te zijn waarbij we de meeste ondersteuning koneden geven.
De kennis die de roostermakers over reiskosten en continuiteit van de personeelsbezetting bezitten, was belangrijk voor de te ontwikkelen kennisbanken. Daarnaast waren er een aantal vaste randvoorwaarden te onderscheiden, die de personeelsplanning niet mag schenden.

Hoofdstuk zes

De kennisstructurering

In de fase van de kennisstructurering hebben we bekeken welke verbanden binnen het kennisdomein kunnen worden onderscheiden. We kiezen in deze fase voor een bepaalde functionele kennisrepresentatie, die het resultaat is van het vertalen van de kennis volgens een bepaald formalisme. De aard van dit formalisme is bepalend voor de vorm waarin de kennis gerepresenteerd wordt.

De opbouw van Nexpert Object

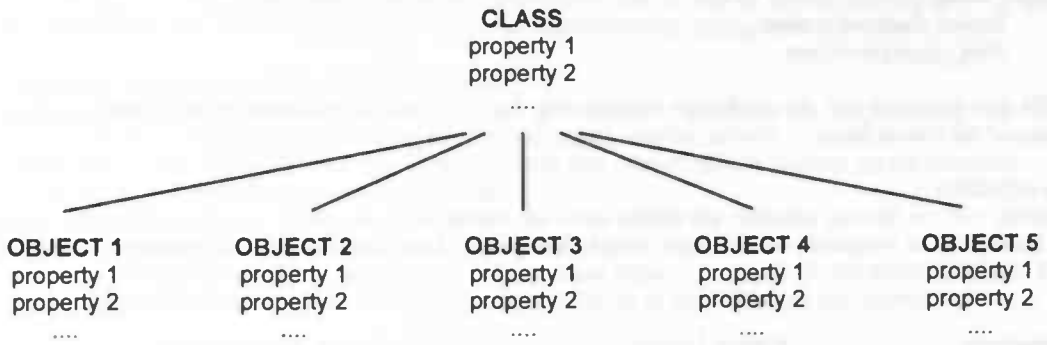
Omdat we bij het realiseren van het kennissysteem de werking en opbouw van de expert-shell Nexpert Object willen gebruiken, hangt de keuze van de kennisrepresentatievorm ook nauw samen met de werking en opbouw van Nexpert. Vaak blijkt dat de controleur en evaluator geen redeneermechanisme nodig hebben en daarom makkelijker door SQL queries geïmplementeerd kunnen worden. Het is voor de duidelijkheid echter goed ze toch op een bepaalde manier te structureren.

De regelstructuur

De expertshell Nexpert heeft het vermogen een systeem van regels en een sterke, objectgeörienteerde representatie te onderhouden. We spreken daarom van een hybride systeem omdat het twee structuren heeft.
De regelstructuur zagen we in het vorige hoofdstuk al kort behandeld. Domeinkennis wordt uitgedrukt in conditie-actie statements, die in de kennisbank worden opgeslagen als regels. Een regel heeft twee delen. Het eerste deel bevat één of meerdere IF-condities die verifieerbaar zijn en moeten voldoen wil het tweede deel, één of meerdere DO-clausules, uitgevoerd worden door het gevolgtrekkingsmechanisme van Nexpert.

De objectstructuur

Objecten beschrijven de variabelen in de kennisbank. Hiërarchische relaties tussen objecten geven grotere redeneerflexibiliteit aan regels door het bestaan van overerving. Daarom bestaan er properties, classes en objecten die de volgende structuur hebben:



We zien dat door de overerving alle objecten ook de properties van de class, waartoe ze behoren, bezitten.
Bij de opbouw van de database van PlanMate zagen we al wat we aan objecten en classes kunnen verwachten in Nexpert.

Welke kennisrepresentatievorm kiezen we?

Er zijn een aantal manieren om kennis te structureren en te presenteren. Een aantal bekende voorbeelden daarvan zijn:

- De propositie- en 1e orde Predikaten Logica
- Semantische netwerken en kennisgrafen
- Produktieregels
- Scripts, Frames en objecten

Over elk van deze mogelijkheden zijn boeken volgeschreven, maar we zagen al bij de beschrijving van Nexpert dat de keuze niet moeilijk is.
Nexpert ondersteunt produktieregels en frames doordat haar structuren vrijwel overeenkomen met deze twee. Het is daarom duidelijk dat we de structurering van de kennis in deze twee representatievormen gieten en dat waren eerlijk gezegd ook de manieren die mij het meest aanspraken.

De structurering

De produktieregels
We beginnen met het structureren van de produktieregels omdat we daarmee kunnen bepalen welke slots we nodig hebben bij de verschillende frames. De informele produktieregels die uit de kennisacquisitie volgen heb ik opgenomen in bijlage B. Ik zal de belangrijkste regels als voorbeeld hier meer structureren zoals ze straks gebruikt gaan worden bij de implementatie. De regels die door Jan Glaudé uit de CAO zijn gehaald hebben we samen bekeken en de meest bruikbare uitgezocht:
#objecten(A) = aantal objecten in verzameling A
A.discipline = de waardes van de disciplines van alle objecten in A
aanbodpool = het personeelsaanbod (de vrije werknemers)
vraagpool = de vraag van de bewerkingen

```
IF      >       #objecten(aanbodpool)     0                    { Als aantal objecten in aanbodpool > 0 }
        =       aanbodpool.discipline     vraagpool.discipline { en discipline van aanbod en vraag gelijk }
THEN    Inroosteren_mogelijk                                   { dan kunnen we diegene gaan inroosteren }
DO      Bepaal_minimale_reisafstand                            { die de kleinste reisafstand heeft }

IF      >       #objecten(vraagpool)      #objecten(aanbodpool)
        =       #objecten(aanbodpool)     0
THEN    Inlenen
DO      Show    "Je moet werknemers inlenen"

IF      =       ploeg.cursus              "EHBO"
        =       #objecten(ploeg)          0
THEN    Niet_Genoeg_EHBO
DO      Show    "Geef EHBO cursus"

IF      >       #objecten(aanbodpool)     #objecten(vraagpool)
THEN    Ontslagen_nodig
DO      Yes     Bepaal_dienstverbanden
        Yes     Kies_ontslagtermijnen
```

Deze regels zijn een gedeelte van de voorlopige modellering. De uiteindelijke modellering liep parallel met de implementatie in Nexpert en Visual Basic.

De frames en objecten
Bij het modelleren van de frames bepalen we welke slots elk frame bezit en geven we dus aan welke waardes bij welke frames horen. Uit de acquisitie werden een aantal slots gefilterd die volgens ons nodig waren voor PlanMate II. Een voorbeeld hiervan:

Frame	werknemers
slot 1	is_een: naam
slot 2	is_een: discipline
slot 3	is_een: postcode

Frame	naam
slot 1	is_een: string

Frame	discipline
slot 1	heeft: "timmerman"
slot 2	heeft: "metselaar"

Frame	string
slot 1	heeft: a .. z
slot 2	heeft: A .. Z

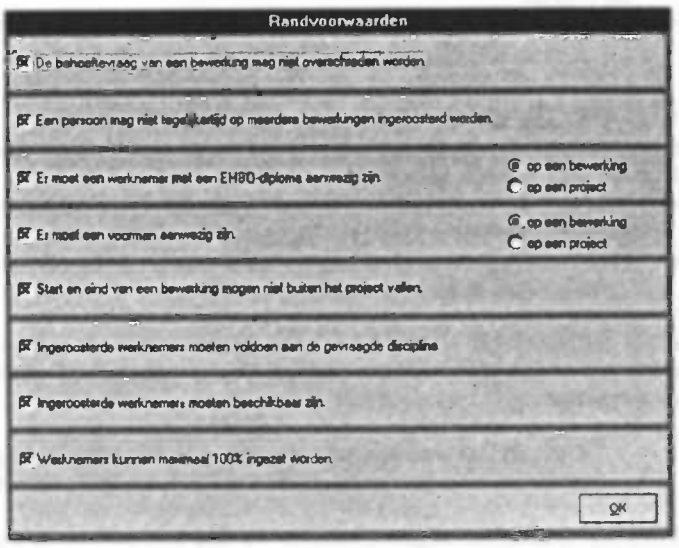
Ook deze modellering had z'n uiteindelijke versie parallel aan de implementatie in Nexpert.
In feite spreken we hier van de database onder PlanMate, zoals die in deel 1 al duidelijk gemaakt werd.

Hoofdstuk zeven

Controleur en evaluator

De controle van de randvoorwaarden

In PlanMate werd de controleur pas geïnstalleerd op het moment dat er vanuit de verkopershoek een aantal eisen aan het pakket gesteld waren. Daardoor is er een kleine toename van randvoorwaarden geweest. Dit verklaart het bestaan van randvoorwaarden die niet uit de kenniselicitatie zijn voortgekomen. Voor de volledigheid zal ik ze echter allemaal behandelen, omdat de toevoegingen ook binnen de eerdere versies van PlanMate passen.
Ik zal beknopt in stappen uitleggen hoe de routines in PlanMate werken. Deze stappen kunnen in de programmacode door middel van SQL-statements uitgevoerd worden, met behulp van de eerdergenoemde Data Definition Language van Visual Basic.



Figuur 7 : Controleur PlanMate

De behoeftevraag mag niet overschreden worden

1. Selecteer bewerkingen die een tijdsperiode binnen de gedefinieerde periode bezitten.
2. Bepaal per bewerking de behoeftevraag, door de behoefte met de tijdsperiode te vermenigvuldigen
3. Bepaal per bewerking de ingeroosterde werknemers en trek dit van de behoeftevraag af.
4. Per bewerking kan zo een overschrijding van de behoeftevraag gesignaleerd worden.

Een persoon mag niet dubbelgeroosterd worden

1. Controleer per persoon
2. Selecteer de roosterstroken van de persoon in de gedefinieerde periode.
3. Controleer op elk beginpunt van een strook, of er al een andere strook bestaat op dat moment.
4. Als dit zo is, is de persoon dubbel ingeroosterd.

EHBO moet aanwezig zijn

1. Selecteer alle personen die op een project of bewerking ingeroosterd zijn en die EHBO'er zijn.
2. Wanneer het resultaat van de selectie groter dan nul is, is er dus minstens één aanwezig.

Voorman moet aanwezig zijn

1. Selecteer alle personen die op een project of bewerking ingeroosterd zijn en die voorman zijn.
2. Wanneer het resultaat van de selectie groter dan nul is, is er dus minstens één aanwezig.

Start en eind moeten kloppen

1. Selecteer alle stroken in de bepaalde periode
2. Wanneer er een periode bestaat waarbinnen deze stroken moeten vallen, zijn dit deelstroken (bewerkingstroken moeten bijvoorbeeld binnen de periode van een project vallen).
3. Selecteer per deelstrook, via het unieke nummer, de strook van de periode waarbinnen de deelstrook moet vallen, door de start van die strook gelijk of eerder te eisen en het eind gelijk of later te eisen.
4. Wanneer deze selectie, meer dan nul records oplevert, valt de deelstrook binnen een periode.

Werknemers moeten de gevraagde discipline bezitten

1. Selecteer per bewerking de personen die op die bewerking geroosterd zijn.
2. Controleer per werknemer of hij de discipline bezit, die de bewerking ook heeft.

Werknemers moeten beschikbaar zijn (PlanMate 0.3)
1. Neem per werknemer de vrije periodes.
2. Selecteer per periode alle roosterstroken van die werknemer, die (gedeeltelijk) binnen deze periode vallen.
3. Wanneer dit steeds nul records oplevert, is de persoon niet foutief ingeroosterd.

Werknemers kunnen maximaal 100% ingezet worden (PlanMate 0.3)
1. Controleer per persoon.
2. Neem uit de 'procenten' tabel alle records van de persoon.
3. Tel voor elke periode de procentuele inroostering van de persoon op in een array die de gedefinieerde tijdsperiode omvat.
4. Controleer of waarden in de array boven de 100% uitkomen.

De evaluatie van de doelfuncties

Bij dit onderdeel moeten we een aantal aannames maken, die misschien niet voor iedereen even logisch lijken. We hebben twee hoofdproblemen:
- Wat zijn de maximale en minimale waarden in de evaluator? Deze hebben we namelijk nodig om de scores in te kunnen schalen.
- Hoe omschrijven we het begrip continuïteit? Dit is een belangrijk begrip in de (personeels)planning en kan op verschillende manieren uitgelegd worden.

Ik zal per onderdeel in de evaluator aangeven hoe ik de waarden bepaald heb, zodat duidelijk is wat de maximale en minimale scores zijn. Nu rest nog het begrip continuïteit, zoals dat het beste toegepast kan worden op een personeelsrooster:

> *Continuïteit is wanneer een waarde, zonder onderbrekingen, vrijwel niet schommelt rond een bepaalde constante. Hoe groter de schommeling rond de constante, hoe lager de continuïteit.*

Wanneer we hiervan uitgaan, moeten we dus een constante bepalen, om de continuïteit te kunnen berekenen en een maximale afwijking om de schommeling procentueel te kunnen vaststellen.

Reiskosten minimaal
Uit de CAO blijkt dat boven de 3.5 uur reistijd, de reistijd van de werktijd afgaat. We kunnen dit dus als maximum aannemen. Een uur reistijd is volgens de CAO 50km, dus het maximum is 175km per dag. Het minimum is 30km per dag, omdat de werkgever geen reiskosten hoeft te betalen aan werknemers die binnen 15km van het werk wonen.
Het uiteindelijke maximum en minimum vermenigvuldigen we nu met de kilometervergoeding.
1. Bepaal per persoon per dag de route (uit de tabel 'route'), die gereden wordt, en bepaal de score op de bovengenoemde schaal, door de kilometervergoedingen te bepalen.
2. Scores die boven de 100% uitkomen moeten zwaarder gewogen worden, door alle kilometers boven de 175 met de factor uurloon/50km te verhogen.
3. Neem het gemiddelde uit alle scores en bepaal zo de uiteindelijke score.

Continuïteit werknemers
Om te beginnen is de constante 100%, en de maximale afwijking 50%. De afwijking is vastgesteld op advies van de roostermakers.
De maximale waarde (en dus een maximale continuïteit) is een afwijking van 0%, en de minimale waarde is als de afwijking gemiddeld 50% is.
1. Neem per persoon de procentuele inroostering per dag.
2. Kijk per dag naar de afwijking met de vorige dag ten opzichte van de constante.
3. Neem het gemiddelde per persoon.
4. Bepaal dit voor alle personen en het gemiddelde is de score.

Pieken en dalen
De constante factor is hier het personeelsaanbod. De maximale afwijking is 10% van het personeelsbestand. De maximale continuïteit is weer 0% en het minimum is een gemiddelde afwijking van 10%
1. Bepaal voor elke dag de vraag naar personeel per discipline
2. Neem de afwijking met de voorgaande dag ten opzichte van het personeelsaanbod per discipline.
3. Het gemiddelde van de disciplines, geeft de score.

Conclusie

Duidelijk is dat de ondersteuning die PlanMate op dit moment kan bieden op het gebied van kennistechnologie al uitgebreid genoemd kan worden. Ook de reacties van de roostermakers op deze mogelijkheden waren zeer positief en bieden perspectief voor de toekomst.
De controleur en de evaluator zijn goed (en snel) te implementeren met SQL en zelfgeschreven routines. Bij het evalueren komen een aantal problemen naar boven waar nu oplossingen voor gevonden zijn, die misschien niet voldoen op het moment dat de evaluator in het echt gebruikt gaat worden. Dit moet de toekomst echter leren.

Deel 3

De ontwikkeling
van het
gebruikersinterface

Inleiding

Het ontwerpen van gebruikersinterfaces is een steeds belangrijker aandachtspunt bij de ontwikkeling van een softwarepakket aan het worden. Na de introductie van Windows onder MS-DOS door Microsoft is het nu eindelijk tot de hele wereld doorgedrongen dat een makkelijk hanteerbaar systeem minder irritatie veroorzaakt, daardoor prettiger werkt, minder gebruikersfouten veroorzaakt en dus beter is.
Daarbij komt dat, in een poging nieuwe, grotere en dus meer winstgevende markten aan te boren, de softwarefabrikanten zich steeds meer gaan richten op de grote groep minder ervaren computergebruikers. Deze groep is een stuk makkelijker te overtuigen van het feit dat een programma goed moet zijn als de visuele vormgeving van de applicatie goed verzorgd is. Dat het functioneren van de rest van het pakket daarbij naar de achtergrond verdwijnt is vaak een welkome bijkomstigheid.
Op de huidige softwaremarkt is het echter steeds meer nodig een makkelijk hanteerbaar pakket aan te bieden, omdat dat steeds meer de maatstaf wordt aan de hand waarvan een pakket door de klant beoordeeld wordt. Een fantastisch programma met een beroerd interface ligt sneller in de prullenbak dan een fantastisch interface op een beroerd stuk software. Ik zal in hoofdstuk acht meer aandacht besteden aan de vraag waarom gebruikersinterfaces nodig zijn en wat een goed interface kenmerkt.
Tot voor kort was er een behoorlijke dosis kennis van C nodig om een windows applicatie te programmeren en dan was het nog maar de vraag of het resultaat bevredigend was. Daar is gelukkig verandering in gekomen met de introductie van Visual Basic (en andere 'visuele' programmeertalen). Door deze programmeeromgeving werd het een stuk eenvoudiger de mogelijkheden van MS-windows te gebruiken in eigen applicaties. In hoofdstuk negen zal ik dieper ingaan op Visual Basic en mijn eigen ervaringen met deze taal beschrijven.
Tijdens de ontwikkeling van PlanMate II werd ook al snel duidelijk dat het nodig was een makkelijk interface te bouwen. Als onervaren programmeur heb ik dit geprobeerd aan de hand van de kennisacquisitie en de ter beschikking gestelde rapportage.
In hoofdstuk tien zal ik duidelijk proberen te maken waarom ik bepaalde zaken op die manier weergegeven heb. Hier zullen een aantal behoeften kort behandeld worden. Voor een uitgebreidere behoeftenanalyse verwijs ik naar het informele behoeftenmodel (Glaudé/van der Meer 1994), wat in deel 1 kort behandeld is en het behoeftenmodel van de modelleringsdocumenten van PlanMate II (van der Meer 1994).
Daarnaast zijn er nog een aantal zaken die 'onder' het interface plaatsvinden. Deze bespreek ik in hoofdstuk elf.
Eén van de belangrijkste redenen dat Nexpert moeilijk te koppelen is is de traagheid van het pakket, en dat brengt ons bij de algoritmes die nodig zijn om een personeelsrooster te genereren. De belangrijkste zal ik in hoofdstuk twaalf behandelen: het optimaliseren van de reisafstanden van de werknemers.

Hoofdstuk acht

Gebruikersinterfaces

Gebruikersinterfaces worden steeds belangrijker gevonden en dus is het een aandachtspunt binnen de software engineering dat steeds meer aandacht krijgt. Wanneer we een gebruikersinterface willen ontwikkelen dat goed aansluit op de manier van plannen van de roostermakers, is een goede kenniselicitatie nodig, maar ook een consistent en gemakkelijk te bedienen planningsscherm.

Waarom een gebruikersinterface?

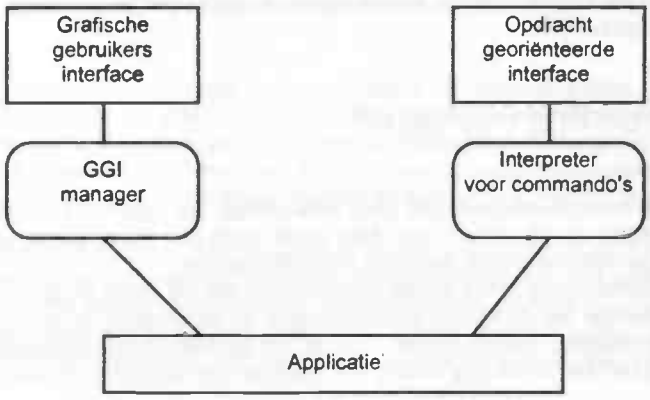
Mens-Computer interactie
De ontwikkeling van de huidige computertechnologie gaat razendsnel. De machines worden groter en sneller en er komen steeds meer mogelijkheden om te werken met data in de vorm van tekst, geluid en beelden in een omvang waarvan men nog niet zo lang geleden dacht dat dat nooit mogelijk zou zijn.
Door de snelheid van deze ontwikkelingen is het vrijwel niet mogelijk dat de mens in datzelfde tempo mee ontwikkelt. Het verschil tussen de groep mensen die de laatste ontwikkelingen en technieken op de voet volgt en de groepen die dit in mindere mate doen wordt daardoor steeds groter. Toch is het tegenwoordig noodzakelijk dat iedereen om moet kunnen gaan met deze technieken en is het dus nodig het gebruik van software en computers zo veel mogelijk te scheiden van de techniek.
Een gebruiker van een softwarepakket moet niet lastiggevallen worden met problemen die samenhangen met het platform waarop gewerkt wordt. Sterker nog, een gebruiker wil hier vaak niet mee lastig gevallen worden en dat hoeft ook niet.
Het is daarom belangrijk dat een gebruiker goed overweg kan met de software. De interactie tussen computer en mens moet dus soepel verlopen.
Het is duidelijk dat hier veel onderzoek naar verricht wordt en dat dat onderzoek altijd achter zal lopen op de laatste technische ontwikkelingen, maar een goed samenspel tussen computer en mens is absoluut noodzakelijk in de huidige softwareontwikkeling.

Het belang van gebruikersgemak
Eén van de best bewezen bevindingen in geheugenonderzoek is dat we zaken veel beter herkennen dan dat we het ons kunnen herinneren. Het overwicht dat herkenning heeft over herinnering heeft uiteraard een grote invloed op het ontwerp van interfaces, en dus ook op het ontwerp van grafische interfaces van software pakketten.
We zien dit terug in de interfaces van deze tijd. Het scherm is gevuld met menu's en icoontjes, waar de gebruiker langs kan gaan tot hij het item herkent dat nodig is om een bepaalde taak uitgevoerd te zien. Bij goede (intuïtieve) interfaces hoeft de gebruiker zich niet meer druk te maken over wat hij precies aan het doen is en gedaan heeft of welke nj commando's hij moet invoeren. Hij moet weten hoe hij menu's en iconen moet selecteren en kan het herinneren overlaten aan het interface, terwijl hij zelf alleen hoeft te herkennen.
Wanneer een gebruiker nog ongeoefend is, is dit een zeer plezierige manier van werken. Mits de semantiek van menu's goed is gekozen, kan de gebruiker deze makkelijk associëren met de actie die hij wil ondernemen en de juiste optie kiezen.
Het is duidelijk dat de gebruiker zal kiezen voor een gebruikersvriendelijk programma en daar de klant koning is, is dit (eindelijk) een steeds belangrijker onderdeel aan het worden in het ontwikkelen van software.

Opdrachtgeoriënteerde gebruikersinterfaces
Bij opdrachtgeoriënteerde interfaces moeten we bijvoorbeeld denken aan UNIX of MS-DOS waar de gebruiker een commando geeft door een tekst in te toetsen. Het commando kan dan vaak wel weer een ander systeem aanroepen of een reeks andere commando's.
Dit heeft bijvoorbeeld als voordeel dat er geen duur kleurenscherm nodig is met een hoge grafische resolutie en dat het interface dus ook vrij eenvoudig te ontwikkelen is. Door de tekstuele invoer is combinatie van commando's vaak een krachtige eigenschap en het vele typewerk is meestal ook wel te verhelpen door de commando's kort te houden.
Nadelige eigenschappen zijn, dat de gebruiker wel veel commando's zal moeten leren en die hoeveelheid kan behoorlijk oplopen. Daarnaast zullen er meerdere verschillende soorten fouten door de gebruiker gemaakt worden. Wanneer je gericht hulp wilt bieden bij een foute invoer, zal er dus een groter systeem ontwikkeld moeten worden om deze hulp te bieden. Ook het niet kunnen gebruiken van de muis en andere soorten hulpmiddelen zorgt voor bepaalde beperkingen.

Grafische gebruikersinterfaces
Tegenwoordig zijn grafische gebruikersinterfaces helemaal in. Dit is niet alleen omdat ze er beter uitzien dan de interfaces van vroeger, maar ook omdat ze voor onervaren gebruikers veel makkelijker te begrijpen zijn. De interfaces

bestaan meestal uit meerdere windows met verschillende informatie en iconen die bepaalde processen of bestanden representeren. Deze grafische ondersteuning gaat echter ook gepaard met een tekstuele weergave, wat de duidelijkheid vergroot. De selectie van commando's vind plaats door middel van menu's en deze zijn weer makkelijk aan te sturen door middel van de muis.
Door het gebruikersgemak is het prettiger werken voor onervaren computergebruikers, maar ook voor ervaren gebruikers is er veel meer overzicht door het gebruik van verschillende vensters. Door die vensters is scherminteractie ook stukken makkelijker geworden.



Figuur 8 : Koppeling interface en applicatie

Natuurlijk sluiten beide interfaces elkaar niet uit en Figuur 8 geeft aan dat een systeem dat beide bevat ook heel goed mogelijk is. De ontwikkeling hiervan is echter wel veel duurder omdat beide interfaces weinig overlap vertonen in ontwikkeling [Draper, 1988].

Conclusie
Het kan voorkomen dat gebruikers kiezen voor opdrachtgeoriënteerde gebruikersinterfaces omdat hier de combinatie van commando's mogelijk is en de interactie met de computer sneller verloopt. het is echter duidelijk dat de grafische gebruikersinterface steeds meer de overhand gaat krijgen. Daarbij was een zekere standaardisatie nodig, en die is er nu ook.
Om aan de ervaren gebruikers tegemoet te komen is het nodig in de grafische interfaces ook besturing via het toetsenbord mogelijk te maken door middel van 'short cut keys', zodat de grafische opbouw van het scherm op den duur niet remmend en irriterend gaat werken.

Ontwerpprincipes bij gebruikersinterfaces

Grafische gebruikersinterfaces hebben intussen dus voor een revolutie gezorgd in de micro-computerindustrie. Daar waar vroeger het cryptische (en voor sommigen angstwekkende) DOS-promptje C:\ op het scherm verscheen, wordt er nu een Microsoft-wereld vol menu's en icoontjes op het scherm getoverd die een goed overzicht geven over wat de computer allemaal te bieden heeft.

Waarom consistentie in gebruikersinterfaces?
Het grotere overzicht bij grafische interfaces geeft de gebruiker de mogelijkheid zich meer te richten op de functionaliteit van de software en hoeft zich daarom minder met de besturing van het programma bezig te houden. Dit verkleint dus ook de kans op fouten. Wanneer er echter geen consistentie in de verschillende interfaces zit, ben je in feite weer terug bij af. De gebruiker moet zich dan namelijk weer meer gaan richten op de verschillende interfaceitems om er achter te komen welke functionaliteit achter de verschillende menu's en knoppen zit.
Ook consistentie in de communicatie tussen de gebruiker en het interface van de applicatie is echter belangrijk zodat de gebruiker snel doorheeft met wat voor een soort commando of window hij te maken heeft.
Het makkelijker toegankelijk maken van de mogelijkheden die de computer biedt brengt dan zelfs gevaren mee, die voor de gebruiker zeer onplezierig kunnen uitvallen. Zo zou je je kunnen voorstellen dat in het ene gebruikersinterface een toetscombinatie 'control+V' betekent dat een object veranderd moet worden, terwijl diezelfde toetscombinatie in een andere applicatie het verwijderen van een object inhoudt. De gebruiker denkt snel en handig met windows om te kunnen gaan, maar snijdt zichzelf aan de slordigheid van de ontwerper.
Consistentie is dus zeker iets dat een ontwerper van een gebruikersinterface in de gaten moet houden.

Waar moet verder rekening mee gehouden worden?
Bij het ontwerpen van gebruikersinterfaces moet men rekening houden met de behoeften, de ervaringen en de mogelijkheden van de gebruiker van het systeem. De gebruikers moeten betrokken worden bij het ontwerpproces en door middel van prototyping moet het commentaar van de gebruikers verwerkt worden in het interface.
Dit wordt lastiger wanneer een gebruikersinterface bedoeld is voor een zeer grote groep gebruikers. Dan kan door middel van een representatieve steekproef veel informatie gewonnen worden en kunnen sommige handelingen zeer voorspelbaar en bepaalde concepten ingeburgerd blijken, zodat deze gebruikt moeten worden in het ontwerp.
Wanneer er geen rekening wordt gehouden met de behoeften en ervaringen van de potentiële gebruikers zal de gebruiker zich tot een bepaald niveau willen aanpassen aan het pakket, maar het is niet verstandig om daarvan uit te gaan.

Hoe maken we het de gebruiker zo makkelijk mogelijk?
Gebruikers maken altijd vergissingen wanneer ze een systeem gebruiken. Zoals we al eerder zagen kunnen we door een goed ontwerp deze fouten minimaliseren, maar het voorkomen van vergissingen is vrijwel niet mogelijk. Het is daarom noodzakelijk dat het interface opties bevat om bepaalde acties te herstellen. Dit kan op twee manieren [Sommerville, 1992]:

1. Het vragen om bevestiging van gevaarlijke acties, zodat de gebruiker er een extra keer op gewezen wordt dat hij iets wil doen dat gevaarlijk kan zijn.
2. Het opnemen van herstelfaciliteiten die het systeem terugbrengen in de toestand die bestond vóór de opdracht werd gegeven. Dit is overigens een bewerkelijke en dus kostbare zaak om te implementeren. Zeker wanneer er meerdere acties plaatsvinden die ongedaan moeten kunnen worden gemaakt.

Daarnaast is het goed wanneer er allerlei hulpfaciliteiten zijn ingebouwd in het interface, waarin ook een zodanige structuur is aangebracht dat de gebruiker zich makkelijk een weg weet te vinden door deze adviezen. Het is daarbij van belang dat de hulpboodschappen en waarschuwingen correct, positief, consistent en constructief zijn zodat de gebruiker op een moment van irritatie niet nog meer geïrriteerd raakt.

Conclusie
We kunnen in het kort de ontwerpprincipes van gebruikersinterfaces weergeven in de volgende opsomming:

1. De interface moet consistent zijn.
2. De gebruiker mag niet door het systeem verrast worden
3. De interface moet gebruik maken van de termen en concepten waarmee de potentiële gebruiker bekend is.
4. Er moet een mechanisme aanwezig zijn waarmee fouten hersteld of voorkomen kunnen worden.
5. De interface moet één of andere vorm van ondersteuning voor de gebruiker bevatten.

Het is belangrijk dat deze aandachtspunten door de ontwerper altijd meegenomen worden in het ontwerp van een gebruikersinterface.

Prototyping van gebruikersinterfaces

Wat is prototyping?
In de literatuur [Sommerville 1992] wordt bij softwareprototyping gesproken over een aanpak van verkennend programmeren, waarbij de eerste fase van de bouw het ontwikkelen van een programma omvat, waarmee de gebruiker kan experimenteren. Het voornaamste doel hiervan is echter door middel van de verkregen terugkoppeling van de gebruikers de systeemeisen vast te leggen. Daarna wordt een herimplementatie van de software uitgevoerd om zodoende een systeem te krijgen dat voldoet aan de gestelde kwaliteitseisen.
Het is dus duidelijk een echt werkend systeem, en geen idee of tekening, maar wel een systeem dat eventueel na gebruik weggegooid zal worden. Het is daarom zaak het snel en goedkoop te kunnen bouwen.

Wanneer kunnen we gebruikersinterfaces prototypen?
Prototyping is alleen mogelijk wanneer we snel en goedkoop een prototype kunnen produceren, omdat anders de ontwikkelkosten van het prototype te zwaar gaan wegen in verhouding met de ontwikkelkosten van het gehele pakket. Wat we dus nodig hebben is een makkelijke ontwikkelomgeving voor gebruikersinterfaces.
Omdat we PlanMate kunnen zien als een commercieel dataprocesprogramma valt de keus al gauw positief uit voor de vierde generatie talen. Het succes van deze talen schuilt vooral in de grote overeenkomsten met gegevensverwerkende applicaties. Doel van veel dataprocesprogramma's is het vervaardigen van rapportage aan de hand van de databases van de onderneming.
Vierde generatie talen zijn vaak opgebouwd rond een databasevoorziening en beschikken over een database vraagtaal, software om rapporten te vervaardigen, een interfacebuilder en enkele spreadsheet-achtige functies. Door de mogelijkheid deze functies makkelijk aan te kunnen sturen is dit een ideale ontwikkelomgeving om prototypes te ontwikkelen. Nadeel is dat niet duidelijk is wat het effect is van deze talen op de kosten gedurende de gehele levenscyclus van een project omdat het gebrek aan standaards op den duur tot onderhoudsproblemen zou kunnen leiden.

Prototyping binnen PlanMate II
We kunnen zeggen dat bij PlanMate II in zoverre gebruik is gemaakt van prototyping dat deze afstudeeropdracht die eerste fase van verkennend programmeren omvatte. Verder zal er in de maanden na deze stage een periode van prototyping zijn waarin de gebruikers mogen experimenteren met het programma.
Bij dit zogenaamde 'task level prototyping' is het dan de bedoeling dat de roostermakers gaan proberen hun roostertaak werkelijk uit te voeren op het prototype. Dit kan nog zo z'n problemen met zich meebrengen omdat het natuurlijk maar helemaal de vraag is of de gebruikers goed kunnen verwoorden wat er volgens hen mis is. Daarbij kan nog komen dat ze zich bij bepaalde wensen niet kunnen voorstellen dat er nog zoiets bestaat als beperkingen aan de hardware of programmeeromgeving waardoor bepaalde zaken niet of nauwelijks realiseerbaar zijn.

Chauffeured prototyping
Er is wel een ronde van 'chauffeured prototyping' geweest tijdens de kennisacquisitie. Tijdens deze sessie keken de toekomstige gebruikers toe, terwijl ik liet zien hoe het systeem functioneerde. Dit was tevens bedoeld als test of het ontwerp van het scherm wel voldeed aan de eisen van de roostermakers. Belangrijker was echter of het planningsconcept dat Jan Glaudé en ik ontwikkeld hadden, aan de hand van de voorgaande gesprekken, overeenkwam met de vraag van de roostermakers. Dit bleek in grote lijnen overeen te komen, wat betekende dat we met deze basis verder konden gaan werken.
De feedback die uit die sessies kwam bleek zeer waardevol, zeker omdat nu visueel zichtbaar was waar over gepraat werd en dat praatte in het vervolg van de kennisacquisitie een stuk makkelijker. Het was achteraf gezien waarschijnlijk een goed idee geweest om na de eerste gesprekken al een gebruikersinterface te ontwikkelen en dit aan de gebruikers voor te leggen. De eerste gesprekken hadden dan wel duidelijker gericht moeten zijn op dat gebruikersinterface. Hier werd duidelijk dat een fase van prototyping op taakniveau waarschijnlijk ook zeer interessant zou zijn.

Waarom prototyping?
Met de opkomst van grafische gebruikersinterfaces, zoals Apple Macintosh, de windows-omgeving voor IBM PC's en compatibles en het X-windows systeem ondersteund door de interface toolkit van Motif, wordt het specificeren, ontwerpen en maken van het gebruikersinterface een steeds belangrijker deel van de ontwikkelingskosten van software.
Prototyping is een belangrijke techniek om systeemeisen vast te stellen en te valideren, en wanneer we naar gebruikersinterfaces kijken, kunnen we zeggen dat prototyping zelfs essentieel is om de systeemeisen op te sporen en dat het ontwikkelen van een papieren ontwikkelingsmodel voor een gebruikersinterface zonder dat dat gecombineerd wordt met prototyping vrijwel zeker tot mislukken gedoemd is.
Nu lijkt het misschien handig dat, door een sterke scheiding te maken tussen de gebruikersinterface en de onderliggende software, het mogelijk is om het interface later nog aan te passen aan de wensen van de klanten. Dit blijkt echter op de huidige softwaremarkt vaak al noodlottig te zijn voor de levensduur van een applicatie en dan is de keus snel gemaakt. Daarbij komt het voordeel dat het prototype als een soort trainingsmedium zal werken waardoor er een gebruiker 'ontstaat' die niet alleen vakkundig is op het vakgebied dat geïmplementeerd moet gaan worden, maar ook vakkundig is in de werking van het systeem wanneer het frequent gebruikt wordt door een gebruiker. Deze kennis is uiteraard zeer belangrijk bij het verder ontwikkelen van de applicatie, omdat de syteemontwikkelaar zelf niet over die kennis beschikt.

Waar moeten we op letten bij prototyping?
Wanneer we een protoype willen laten draaien bij een aantal gebruikers, moeten we op een aantal zaken letten:
- Het is belangrijk van te voren goed duidelijk te stellen wat de aandachtspunten worden tijdens de fase van prototyping, daar ook een plan voor op te stellen en dat aan de gebruiker voor te leggen.
- Er moet ook een plan gemaakt worden hoe de feedback van de gebruiker geëvalueerd zal gaan worden.
- De keuze van de personen die het prototype gaan uitproberen moet zorgvuldig zijn en kan het beste bestaan uit technische en niet technische personen.

Alle veranderingen die gemaakt worden aan de hand van de feedback moeten kort gedocumenteerd worden in verband met hergebruik van deze kennis, maar ook om duidelijk op papier te hebben waarom bepaalde zaken anders bleken te moeten dan de ontwikkelaars dachten.

Conclusie
Binnen een gebrukersinterface is consistentie erg belangrijk. De gebruiker moet daarnaast via allerlei help-functies tijdig gewaarschuwd worden voor fatale fouten.
Door middel van prototyping van het gebruikersinterface kunnen we vervolgens zorgen voor een naadloze aansluiting van de applicatie op de roostertaak.

Hoofdstuk negen

Visual Basic

Programmeren als laatste ontwerpfase

De modellering en de implementatie-omgeving
Gedurende de eerste stappen van het ontwerptraject van een softwarepakket is het verstandig nog niet te kijken naar de implementatieomgeving waarin de uiteindelijke implementatie moet gaan plaatsvinden. Deze kan namelijk nog niet gekozen zijn en afhangen van de eerste stappen van de modellering. Het kan echter ook zo zijn dat de software op meerdere platforms zal moeten gaan draaien, maar het is het meest waarschijnlijk dat de implementatie-omgeving het veranderlijkst is van het hele systeem.
Wanneer het toch onvermijdelijk is bepaalde beperkingen aan het model op te leggen die te maken hebben met de implementatie-omgeving, is het verstandig deze zeer lokaal te houden. Op deze manier kunnen ze lokaal opgelost worden, op het moment dat er veranderingen optreden in de omgeving [Jacobson 1993].

De modellering en de programmeertaal
Ook al is het uiteindelijke programmeren één van de laatste ontwerpfasen, de gekozen programmeertaal beïnvloedt wel degelijk het ontwerp. Daarbij is het het belangrijkst te letten op basiseigenschappen van de taal en z'n omgeving. Bij een objectgeörienteerd ontwerp is het bestaan van bijvoorbeeld overerving van invloed op het ontwerp, maar ook errorhandling en automatische garbage collection moeten wel of niet opgenomen worden in de modellering wanneer de taal ze niet ondersteund.
De keuze van de programmeertaal was bij PlanMate niet vrij omdat KPD Automatisering met Visual Basic werkt en PlanMate bij hen verder ontwikkeld zal worden. Zij zien het pakket daarom graag geschreven worden in Visual Basic, maar er zijn zeker zaken te noemen die spreken voor het gebruik van deze programmeeromgeving. Ik zal een aantal voordelen en nadelen behandelen.

Waarom Visual Basic?

Programmeren onder DOS
Vroeger was het nog niet zo eenvoudig een DOS-applicatie te schrijven. De programmeur moest aan veel te veel dingen denken: Wat deed de muis? Waar was de gebruiker in het menu? Werd er ergens op de muisknop geduwd? Allemaal problemen die voortdurend terugkwamen en een flinke hoeveelheid kennis van de programmeertaal C vereisten, en dan was het nog zo ingewikkeld, dat menig programmeur overspannen raakte.
Het programmeren onder windows maakte dit alleen maar ingewikkelder en wanneer een aantal programmeurs dezelfde consistentie in hun interfaces wilden aanbrengen kwam het er op neer dat ze voortdurend allemaal met dezelfde problemen worstelden, wat natuurlijk totaal onnodig is.

Waarom het gebruikersinterface ontwerpen in Visual Basic?
Visual Basic is daar op in gesprongen en volgens de laatste geruchten zijn er ook andere visuele programmeeromgevingen op komst (of al beschikbaar), zodat we straks in Visual C++ en Visual Pascal kunnen werken.
In Visual Basic zijn al de bovengenoemde, lastige implementatieproblemen in feite al opgelost voor de programmeur en dat maakt het programmeren van windowsapplicaties een stuk aangenamer. Zoals al eerder genoemd is een vierde generatie taal uiterst geschikt voor het implementeren van een gebruikersinterface en Visual Basic heeft veel eigenschappen van zo'n taal. Het is in feite een makkelijke ontwikkelomgeving waarin de omgeving van de applicatie snel neergezet kan worden en kan dus gezien worden als een prima software prototyping tool.
Omdat de tools van Microsoft gebruikt worden, lijkt het geschreven gebruikersinterface op MS-windows en is het alleen zaak voor de programmeur om de functionaliteit die achter de verschillende menu-items zit, consistent te houden met andere windows applicaties.
Een belangrijk punt is dat de snelheid van de applicaties redelijk intact blijft, waardoor het ontwerpen van het gebruikersinterface in Visual Basic al een hoop problemen wegneemt en ook zeer snel kan gebeuren.
Dit is interessant wanneer we door middel van protoyping de gebruikers willen betrekken in het ontwerpen van het softwarepakket. De toekomstige gebruikers kunnen snel zien hoe de programmeur denkt dat hij hun wensen en behoeften moet implementeren en ze kunnen daar snel op reageren en zaken veranderen.

Waarom programmeren in Visual Basic?
Zo op het eerste gezicht schrikt de naam Visual Basic nogal af door de associaties met de eerste vormen van basic die computerfreaks (geboren voor 1974) nog wel kennen. Dit is echter onterecht. Visual Basic heeft veel meer weg van C en Pascal dan van het oude Basic, en bestaat uit Functions en Procedures (Subs) die overigens alleen geactiveerd worden als reactie op events. Het bestaan van Functions en Procedures betekent dat er ook makkelijk van recursie gebruik gemaakt kan worden.
Het is mogelijk types te declareren en je hoeft je geen zorgen te maken over garbage collection. Ook errors kunnen worden waargenomen en waar nodig kan het programma bij een error gestuurd worden in de richting die de programmeur wenst.
Visual Basic staat moderne modulaire programmeertechnieken toe waardoor de programmeur z'n programmatekst kan scheiden van het gebruikersinterface. Dit is een groot voordeel, omdat veranderingen in het gebruikersinterface nu geen effect hoeven te hebben op de achterliggende code. Hierdoor zijn op- en aanmerkingen tijdens de prototyping periode makkelijk te implementeren, zonder de programmacode te moeten herschrijven.

Databases in Visual Basic
Bij het project PlanMate bleek al snel dat we spreken over een uitgebreide database waarin makkelijk met gegevens omgesprongen moet kunnen worden. Het is dus interessant wanneer een programmeertaal databasemanagement ondersteunt.
Visual Basic heeft een uitgebreide set commando's om databases en hun inhoud te bewerken. Ik zal hier in hoofdstuk elf uitgebreider op terug komen, maar het is mogelijk databases, tabellen en velden te maken, te verwijderen en te vullen met gegevens , die op hun beurt ook weer verwijderd kunnen worden. Deze databasegegevens kunnen met SQL queries geselecteerd worden, wat de mogelijkheden tot datamanipulatie nog meer vergroot.

De ontwikkelshell
De shell waarin gewerkt wordt bij het programmeren in Visual Basic is zeer uitgebreid. Dit vereist volgens mij overigens wel een monitor met een oplossend vermogen van minimaal 800x600, door de grote hoeveelheid aan windows en hulpmiddelen.
Het menu is compleet te noemen en de edit en debug mogelijkheden zijn goed. De uitgebreide help-functie is zeer nuttig wanneer bepaalde zaken onduidelijk blijken. Bij ieder window dat je toevoegt aan de applicatie (in Visual Basic heet dit een 'form') is het mogelijk een menu te maken door middel van het Menu Design window.
Wanneer we een programma runnen hebben we te maken met een interpreter. Dit geeft de zeer handige mogelijkheid om te pauzeren en dingen te veranderen, of waardes van variabelen op te vragen. Wanneer we een executable willen creëren is dit ook mogelijk en krijgen we een versie van het programma die onafhankelijk onder windows kan draaien (mits enkele *.dll files aanwezig zijn.)
Deze libraries die nodig zijn voor het laten draaien van de executable zijn uiteraard niet op alle computers aanwezig. Visual Basic heeft daarom een 'setup wizard' waarmee een aantal installatiediskettes aangemaakt kunnen worden, die alle files bevatten die nodig zijn.
Het is jammer dat het maken van backups moeilijk is doordat dan elke module een andere naam gegeven moet worden, maar het heeft ook weer z'n voordelen dat iedere module z'n eigen naam heeft. Je kunt ze namelijk apart creëren en daarna introduceren in het project.
De makkelijkste manier van een backup maken is door alles in een archiveerprogramma in te pakken. De file die nu ontstaat is niet al te groot en is makkelijk weer terug te halen.

De taal Visual Basic

De code van Visual Basic lijkt naast C het meest op Pascal en is dus duidelijk een imperatieve programmeertaal. Dit betekent dat in Visual Basic geschreven code dus stap voor stap wordt uitgevoerd en daarom bestaat uit een lange rij toekenningen en berekeningen van kleine stukjes data. Hierdoor wordt de toestand van de machine voortdurend veranderd. We spreken van programmeren met bijeffect [Kamin 1990].
In tegenstelling tot Pascal bestaat Visual Basic eigenlijk uit een verzameling programma's die de gebruiker kan laten lopen door in het gebruikersinterface een event te activeren. Deze programma's zijn in zoverre onafhankelijk van elkaar dat ze elkaar niet kunnen activeren. Ze kunnen wel dezelfde, door de gebruiker gedefinieerde, functies en procedures aanroepen en globale variabelen wijzigen.
In vergelijking met Pascal speelt recursie in Visual Basic maar een kleine rol omdat er niet van pointers gebruik gemaakt kan worden. Daarom is iteratie verreweg de belangrijkste methode van programmeren. Deze taalstructuur ligt dicht bij de onderliggende machine architectuur waar ook veel gebruik gemaakt wordt van sprongen (in Visual Basic zijn dit o.a.: **goto, if, while** en **loop**) en blokken geheugen om data in op te slaan.
Visual Basic is wat programmeermogelijkheden betreft beperkter dan Pascal, maar heeft een uitgebreide verzameling commando's om makkelijk databases te manipuleren. Zoals eerder gezegd zal ik daar later nog kort op terug komen. Daarnaast onderscheidt Visual Basic zich voornamelijk door de gebruikersvriendelijkheid en de visuele mogelijkheden.

Data types
Visual Basic behandelt data voornamelijk in enkele stukken zoals integers, long integers, doubles en karakters. Ook in arrays worden de waardes één voor één behandeld.
De volgende datatypes zijn in Visual basic aanwezig:

- **String**:	een rij van maximaal 65535 karakters
- **Integers**:	gehele waarde tussen -32.768 en 32.767
- **Long integers**:	gehele waarde tussen -2.147.483.648 en 2.147.483.647
- **Single-precision**:	een waarde van maximaal 38 digits waarvan er vanuit gegaan moet worden dat de eerste 7 digits precies zijn.
- **Double-precision**:	een waarde van maximaal 300 digits waarvan er vanuit gegaan moet worden dat de eerste 16 digits precies zijn.
- **Currency**:	een exacte waarde met maximaal 14 digits links van de komma en 4 rechts van de komma
- **Variant type**:	een type dat alle bovenstaande types mag bevatten. Door middel van conversies wordt automatisch bepaald welk type wordt opgeslagen.

Commando's
Er is natuurlijk een grote hoeveelheid commando's aanwezig, maar omdat in Visual Basic vrijwel alleen maar toekenningen en iteratie plaatsvinden zijn de belangrijkste commando's:

- **For** ... **Next**	herhaal de loop een vast aantal keer
- **While** A ... **Wend**	herhaal de loop zolang A voldoet
- **If** A **Then** B **Else** C	als A geldt doe dan B, ander doe C
- **Do Until** A ... **Loop**	tot A geldt moet de loop herhaald worden
- **Do While** A ... **Loop**	zolang A geldt moet de loop herhaald worden
- **Select Case** A	voer verschillende commando's uit bij verschillende waarden van A

Operaties
De belangrijkste operaties zijn hieronder opgenomen, maar door de vele ingebouwde functies, is dit geen goed beeld van de kracht van Visual Basic. Er zijn datum/tijd-functies, financiële functies, stringfuncties, numerieke functies en conversie functies:

+ , &	als A en B string waarden zijn dan is A + B (of A & B) de string die volgt wanneer de karakters uit A gevolgd worden door de karakters van B.
+	als A en B numerieke waarden zijn dan is A + B de som van A en B
- , * , /	analoog aan + voor numerieke waarden.
= , < , > , <= , >=	geeft True als hun argumenten nummers zijn, en de aangegeven conditie geldt.
AND, OR, &	geeft True als hun argumenten condities zijn, en de aangegeven conditie geldt.
NOT	geeft True als zijn argument een conditie is, en niet geldt.

Door het variante type en een aantal operaties die op verschillende types kunnen worden toegepast, lijkt er polymorphisme in Visual Basic te bestaan. Achter het variante type zitten echter een aantal conversies die bepalen welk type er toegekend is aan de variabele en de variabele wordt vervolgens op die manier geïnitialiseerd. Operaties die op verschillende types lijken te werken roepen in feite verschillende functies aan. Weinig polymorphisme dus.

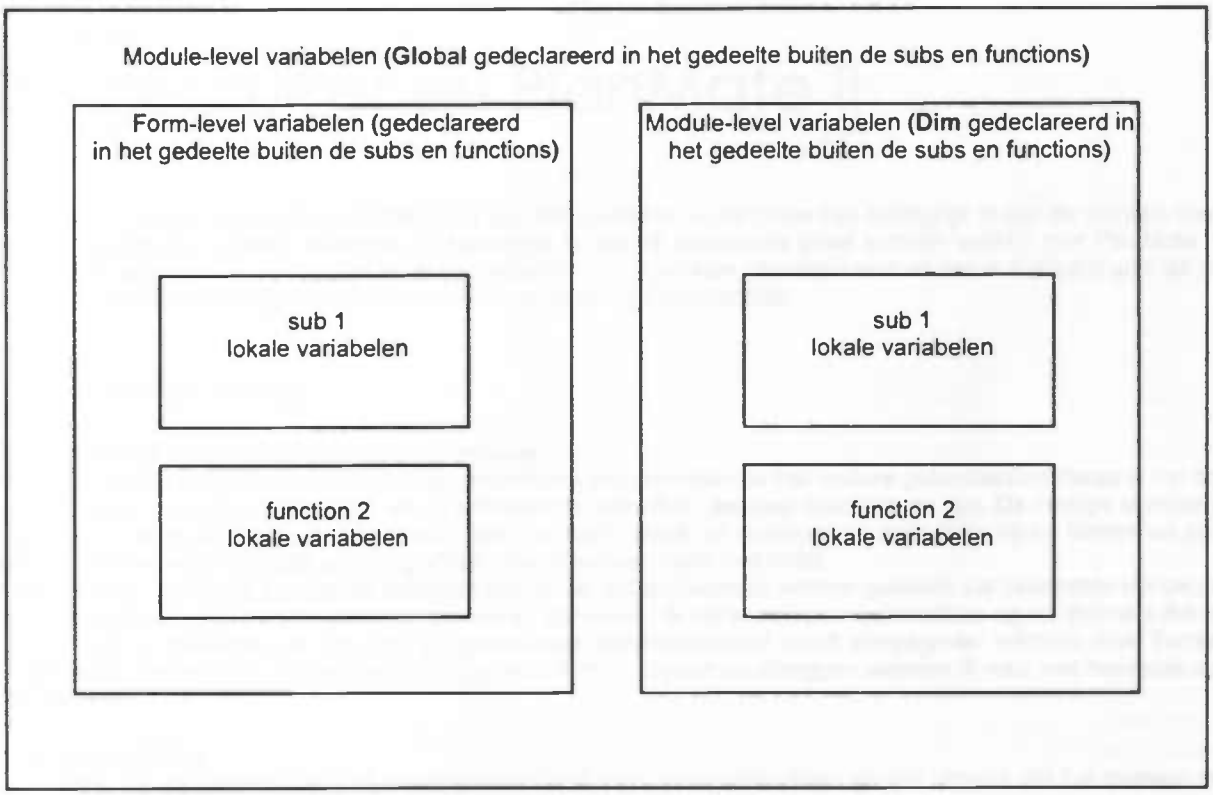
De syntax
In Visual Basic bestaat de mogelijkheid om verschillende modules te maken waarin subs en functions gedefinieerd kunnen worden. Er zijn modules waarin alleen code aangemaakt kan worden en er zijn forms die gekoppeld zijn aan het gebruikersinterface en event subs bevatten waarin lokale subs en functies aangemaakt kunnen worden. Event subs reageren alleen wanneer door een handeling van de gebruiker een zeker event geactiveerd wordt.
Vanuit de codemodules kunnen geen subs en functions uit de forms aangeroepen worden, maar vanuit forms kan wel een sub of function uit een codemodule aangeroepen worden. Er bestaat overloading, dus wanneer we binnen een form een sub aanmaken die al in een code-module staat, wordt binnen de form de sub aangeroepen die in de form gedeclareerd is en kunnen we dus niet meer de globaal gedeclareerde sub aanroepen vanuit die form.

De syntax is hieronder sterk vereenvoudigd weergegeven. Dit geeft een aardig beeld van de opbouw van Visual Basic, maar is in werkelijkheid een klein gedeelte van alle ingebouwde functies die VB bezit.

input	->	formdef | moddef
formdef	->	dimlijst* | subdef | fundef | eventdef
moddef	->	dimlijst* | globallijst* | subdef
		| fundef
dimlijst	->	**dim** variabele **as** type)
globallijst	->	**global** variabele **as** type
staticlijst	->	**static** variabele **as** type
subdef	->	**sub** naam arglijst dimlijst* staticlijst* explijst **end sub**
fundef	->	**function** naam arglijst **as** type dimlijst* staticlijst* explijst **end function**
eventdef	->	**sub** naam_event arglijst dimlijst* staticlijst* explijst **end sub**
arglijst	->	(variabele* **as** type)
explijst	->	(expressie*)
		| (**if** expressie **then** expressie **else** expressie)
		| (**while** expressie explijst **wend**)
		| (**for** variabele value-op value **to** value explijst **next**)
		| (**do until** expressie explijst **loop**)
		enz ..
expressie	->	value
		| variabele value-op variabele
		| **NOT** (expressie)
		| expressie bool-op expressie
value	->	integer
value-op	->	+ | & | - | * | / | = | >= | <= | < | >
bool-op	->	& | AND | OR
variabele	->	naam
naam	->	een rij karakters, zonder leestekens of tekens die gebruikt worden voor de operanden.
event	->	**click** | **dbl-click** | **mouse-move** | .... zo zijn er per object binnen het gebruikersinterface verschillende events. Met een handeling binnen het gebruikersinterface activeren we de eventdef
type	->	gebruikergedinieerd type | bovengenoemde data types

Een duidelijk verschil met Pascal en veel andere talen is dat er geen subs (procedures) en functions gedefinieerd kunnen worden binnen een sub of function-declaratie. Dit ontbreken van geneste functions en subs vereenvoudigt het zoeken naar variabelen binnen de verschillende scopes. Daarentegen is er binnen Visual Basic een andere manier van declareren. Het zoeken binnen de verschillende scopes blijft echter hetzelfde.
Bij variabelen wordt begonnen met het zoeken van de declaratie in de laatst gecreëerde scope en wanneer de variabele daarin niet gedeclareerd is, wordt in de laatst daarvoor gecreëerde (en nog actieve) scope gezocht. Verschil met andere talen is dat er hier in maximaal drie stappen de declaraties van de gezochte variabelen gevonden kunnen worden.
Eerst zoeken we dus binnen de actuele en lokale variabelen en wanneer ze daar niet gedeclareerd zijn kijken we binnen de lokale variabelen van de form of module. Tenslotte zoeken we in de globaal gedeclareerde variabelen van de modules.

Minpuntjes in Visual Basic
Er zijn weinig gebreken in Visual Basic te ontdekken, maar een nadeel vind ik het strakke event-gedreven karakter van de taal. Daardoor is het niet mogelijk berekeningen uit te voeren wanneer de gebruiker even niets in het gebruikersinterface doet.
De taal is ook niet geschikt om veel berekeningen in uit te voeren. Het is echter mogelijk routines in objectfiles aan te roepen, waardoor we deze bijvoorbeeld in pascal of C kunnen schrijven en ze dan vanuit Visual Basic aanroepen.
Ik zou me kunnen voorstellen dat het ontbreken van pointers ook als een nadeel beschouwd wordt, maar door de aard van de applicatie PlanMate ben ik niet zoveel met dit probleem in aanraking geweest.
Uiteindelijk moet een Visual Basic applicatie natuurlijk in de windows omgeving draaien en is het dus erg lastig wanneer er een grafische kaart aanwezig is in de computer waar op gewerkt moet worden, die de vele grafische mogelijkheden van windows niet snel aankan.



Figuur 9 : Verschillende levels binnen Visual Basic.

Conclusie

Visual Basic leek een goede keuze, maar het was de vraag hoe de ondersteuning op de lange termijn zal zijn. Bij KPD is al een applicatie geschreven in Visual Basic en de programmeur die daaraan werkt is tevreden, dus laten we maar op zijn ervaringen afgaan. Dat bewuste pakket werd door mij in het 'behoeftemodel' wel wat traag bevonden, maar ik denk dat dit meer te maken heeft met de programmeertaal zelf dan met de manier waarop de programmatuur geprogrammeerd is.

Hoofdstuk tien

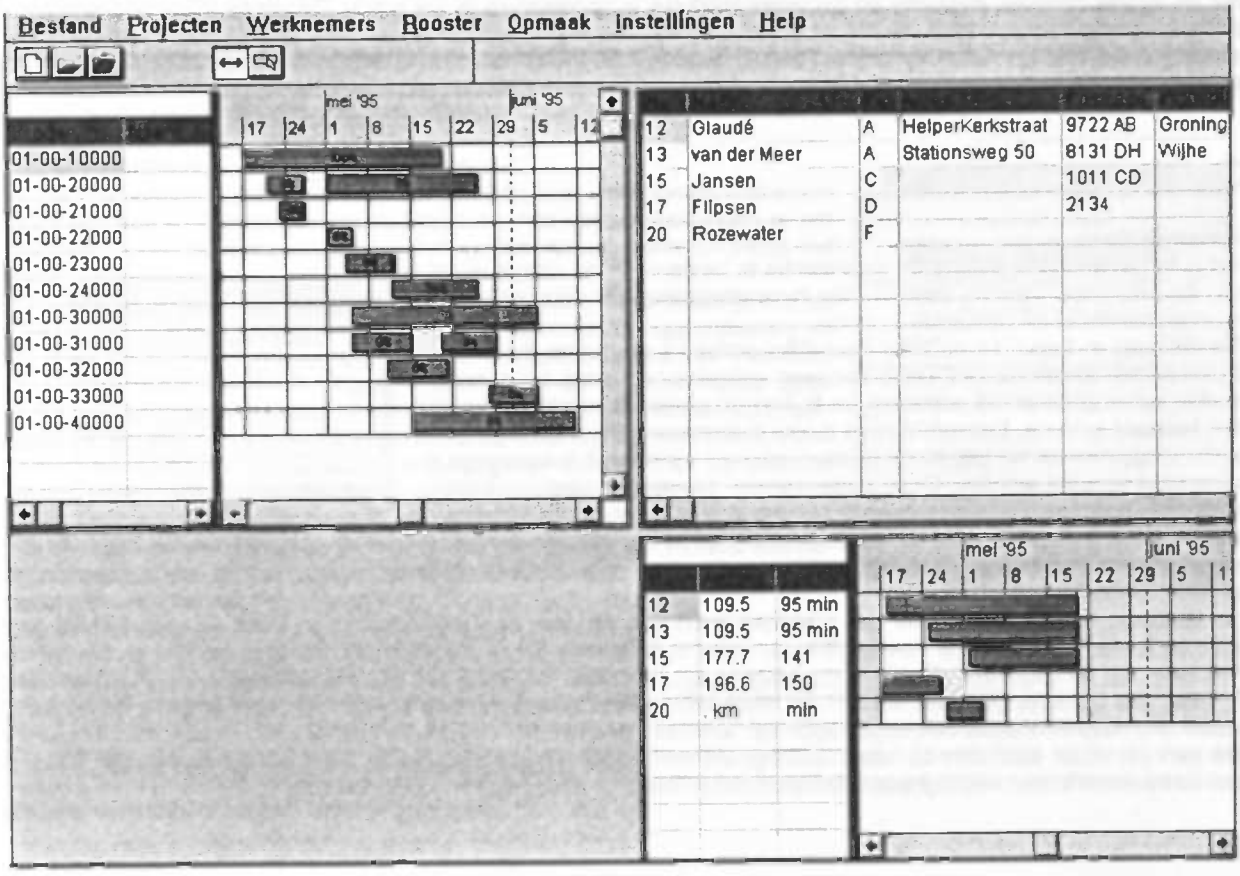
De gebruiker en PlanMate II

We zagen al eerder dat bij de ondersteuning van het roosteren in de bouw het belangrijk is dat de wensen van de gebruikers gehoord worden. Minstens zo belangrijk is dat de gebruikers goed kunnen werken met PlanMate. Het gebruikersinterface is de enige plek waar de gebruiker met PlanMate communiceert en het is dus belangrijk dit zo te ontwerpen dat het makkelijk en duidelijk aansluit op de huidige roostertaak.

Het hoofdscherm

Een consistent model voor gebruikersinterfaces
Wanneer we een gebruikersinterface willen ontwikkelen dat consistent is met andere gebruikersinterfaces is het nodig te kijken naar de huidige standaard, en de entiteiten te gebruiken die daar beschikbaar zijn. De huidige standaard is eigenlijk de interface-stijl die met de Motif-toolkit gemaakt wordt, en wanneer we naar PC's kijken komen we bij Microsoft windows voor MS-DOS uit, dat grafisch vrijwel overeen komt met Motif.
Er zijn in deze standaard een aantal entiteiten die op het scherm kunnen worden getoond. De bekendste zijn de drukknoppen, schakelaars, menu's, panelen, frames en schermen. Ik zal er een paar behandelen op het moment dat ik ze gebruikt heb in PlanMate. In het door mij geschreven behoeftenmodel wordt aangegeven waarom deze bepaalde functionaliteit aanwezig is. Ik zal daar hier nog even kort op ingaan en uitleggen waarom ik voor een bepaalde opzet gekozen heb.

De schermindeling
We moeten bij het ontwerp van het gebruikersinterface voor de roostermakers er van uitgaan dat het overeen moet komen met het bureau van de roostermaker. Daarom is het goed te kijken hoe de roostermakers te werk gaan bij het



Figuur 10 : Hoofdscherm PlanMate 0.3

vervaardigen van een personeelsrooster.
Het uiteindelijke werkblad is het hoofdscherm van PlanMate en is afgebeeld in figuur 10. Bij het ontwerp van dit scherm was het belangrijk op een aantal zaken te letten, en het is dus noodzakelijk er voor te zorgen dat het gebruikersinterface aan die zaken voldoet:

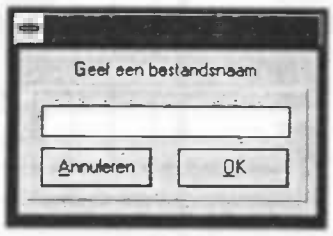
- De roostermaker heeft als basisgegevens de projectgegevens en de werknemersgegevens nodig om een rooster te kunnen maken.
- De projectgegevens kunnen uit PlanCalc komen en zijn daar op een bepaalde manier weergegeven. Om verwarring te voorkomen is het zaak ze in PlanMate ook zo weer te geven.
- De roostergegevens worden bij de bouwbedrijven over het algemeen in strokenschema's gerapporteerd. Dit is duidelijk terug te vinden in de roosterschema's in het document 'Rapportage bij de participerende bouwbedrijven' [van der Meer 1994].

De belangrijkste gegevens zijn de projectplanningsgegevens. Dit bleek uit de procedure die de roostermaker volgde bij het samenstellen van een rooster. Het is dus goed deze duidelijk in beeld te brengen.
Het weergeven van de ingeroosterde werknemers is vervolgens niet belangrijk op het scherm, en is dus daarom in verhouding ook betrekkelijk klein. Op het moment dat deze roostergegevens uitgeprint worden, zijn ze wel erg belangrijk en het is dan dus zaak een overzichtelijk rooster te kunnen uitdraaien.

Het menu
We spreken van een op menu's gebaseerd systeem wanneer de gebruiker kan kiezen uit een aantal geboden mogelijkheden. Voordeel hiervan is dat de gebruiker de opdrachten niet exact hoeft te herinneren omdat er een lijst met opdrachten getoond wordt. De fouten die de gebruiker bij het intypen van een opdracht kan maken worden op deze manier ook afgevangen en wanneer er een foute keuze wordt gemaakt kan er gericht constructieve hulp worden geboden [Preece, 1994].
Het is echter wel moeilijk combinaties van commando's weer te geven in een menu, zonder dat het heel groot wordt. Wanneer dit menu dan te groot wordt zal het zodanig gestructureerd moeten worden dat de gebruiker het toch nog overzichtelijk kan gebruiken. Een menu kan voor ervaren gebruikers soms langzamer zijn dan opdrachtgeoriënteerde interfaces, wat uiteraard irriterend kan werken.
De acties die mogelijk zijn binnen PlanMate zijn opgenomen in het menu. Dit houdt het inladen van gegevens in, het aansturen van rapportagemodules en het maken van bepaalde instellingen in het hoofdscherm. Ook het aansturen van de aparte controle, evaluatie en genereer-modules is opgenomen in het menu. De grootte van de menu's is nu echter nog maar zo gering dat een goede structurering nog niet noodzakelijk is. Dit is uiteraard met het oog op uitbreidingen wel iets om rekening mee te houden, vandaar dat in het hoofdmenu een onderscheid is gemaakt in duidelijk verschillende functies met een ongeveer evengrote hoeveelheid functies als subitems.

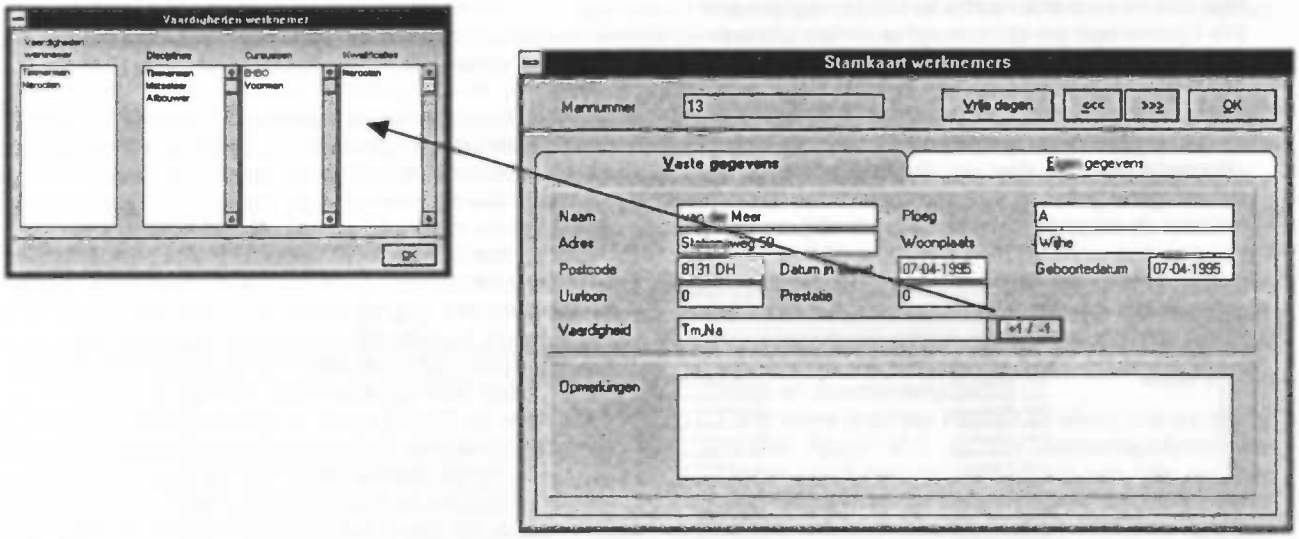
Gegevens invoeren

Invoerwindows
Het is in het interface belangrijk consistentie te bewaren in de communicatie met de gebruiker. Daarom bestaan alle invoerwindows waar tekstuele invoer moet plaatsvinden uit eenzelfde opbouw.
We zien dat in figuur 11 de twee mogelijke acties bij de invoer duidelijk op drukknoppen aanwezig zijn. Deze knoppen gebruiken we vaak voor veelgebruikte acties zoals acties waarmee we te kennen geven dat we klaar zijn met een bepaald scherm. Daarom zijn ze duidelijk zichtbaar. Dit is ook het geval bij de windows waar de gegevens uit de databases verwijderd, toegevoegd en veranderd kunnen worden. Deze zogenaamde 'stamkaart'-windows hebben eenzelfde opbouw. Er is geprobeerd een zo duidelijk mogelijke indeling te maken waarbij de attribuutnaam duidelijk aangegeven is voor de invoerbox waarin de attribuutwaarde staat of moet komen te staan. Figuur 12 geeft de werknemersstamkaart weer.



Figuur 11 : Invoerscherm tekst PlanMate

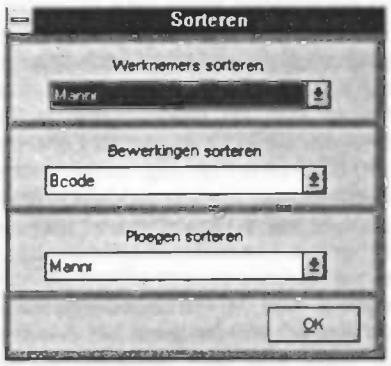
Het textboxje 'opmerkingen' was een duidelijke wens van één van de roostermakers en biedt de mogelijkheid gegevens op te slaan zonder daar aparte attributen voor te definiëren. Deze mogelijkheid draagt in zoverre problemen met zich mee, dat de gegevens die hierin opgeslagen zijn helemaal los staan van alle mogelijkheden die PlanMate biedt. Er moet dus duidelijk gemaakt worden dat deze tekst alleen ingevoerd mag worden wanneer er geen attributen zijn waaronder deze informatie ook opgeslagen kan worden.
We zien dat onder de drukknop naast 'vaardigheid' een ander window verstopt zit. Dit is het window waarin aangegeven kan worden welke vaardigheden een werknemer bezit. Ik zal dit een instelwindow noemen.



Figuur 12 : Werknemersstamkaart PlanMate

De instelwindows voor attribuutwaarden
Voor een aantal zaken is het nodig een instelling te maken. Zo moet aangegeven worden hoe gegevens gesorteerd en geselecteerd moeten worden, maar bijvoorbeeld ook hoelang periodes moeten zijn waarover rapportage moet plaatsvinden en zoals we bij de werknemersstamkaart al zagen welke vaardigheden een werknemer bezit. Onder vaardigheden versta ik hier welke disciplines, cursussen en kwalificaties de werknemer bezit.
Bij het instellen van deze items is het belangrijk dat de roostermaker zo weinig mogelijk zaken hoeft in te typen. Dit voorkomt typefouten.
Bij het zogenaamde instelwindow zijn alleen keuzemogelijkheden aanwezig en kan er dus geen typefout optreden. Het kiezen werkt hier ook sneller dan bij tekst georiënteerde invoer. Dit neemt uiteraard niet weg dat de roostermaker geen vaardigheden kan toekennen aan werknemers die deze vaardigheden helemaal niet bezitten, maar dat spreekt vanzelf en is ook niet te ondersteunen met waarschuwings- of herstelfuncties.

De instelwindows voor datarepresentatie
Doordat de roostermaker te maken zal hebben met werknemersbestanden van meer dan 100 werknemers en projecten met veel bewerkingen is het noodzakelijk dat hij deze bestanden kan selecteren en sorteren zodat het zoeken in de gegevens een stuk aangenamer wordt.
In de behoeftenanalyse die door ons gepleegd werd, werd duidelijk dat de selectie die gepleegd zal gaan worden voornamelijk op vaardigheden en tijdsperiodes zal zijn. Daarom zijn deze in de eerste versie van PlanMate geïmplementeerd. De invoer van de tijdsperiode is op dezelfde manier geregeld als bij het invoerwindow voor een tijdsperiode. Bij het instellen van vaardigheden wordt gebruik gemaakt van de instelwindows die ook bij de stamkaart gebruikt werden. Wanneer een vaardigheid vereist is bij de selectie, worden alle personen geselecteerd die deze vaardigheid bezitten.
Bij het sorteren is de invoer beperkt tot een keuzemogelijkheid. De dropdown lists bevatten de attributen per gegevenstabel en door middel van het selecteren van een attribuut geef je aan hoe er gesorteerd moet worden in een bepaald gegevenspaneel.

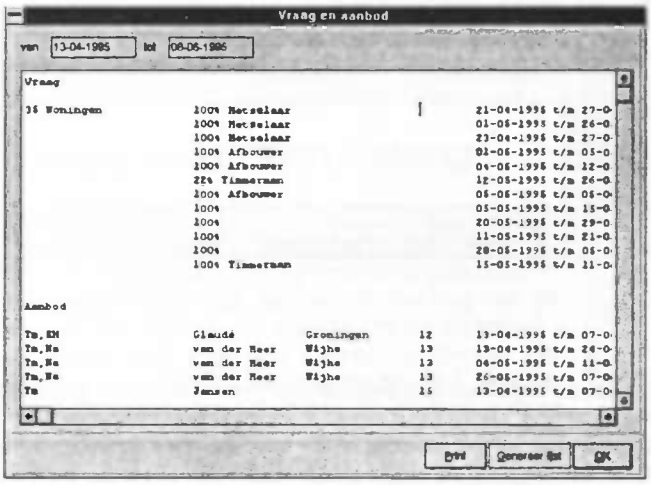


Figuur 13 : Sorteerwindow Planmate

De weergave van informatie in de rapportagemodules

Naast het makkelijk kunnen genereren van personeelsroosters, werd het door de roostermakers minstens zo belangrijk gevonden dat er aan de hand van dit rooster bepaalde informatie uit het systeem te verkrijgen is. Deze rapportage kan een belangrijke ondersteuning zijn bij beslissingen die binnen het bedrijf genomen moeten worden en heeft daarom veel aandacht gekregen binnen PlanMate.

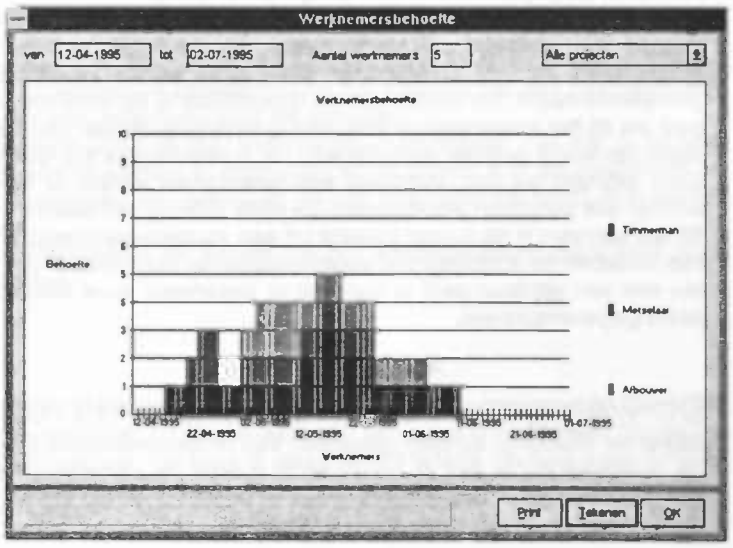
Hoe kunnen we informatie het best weergeven?
We kunnen een aantal vormen van informatiepresentatie onderscheiden die in gebruikersinterfaces voorkomen. Zo is er statische en dynamische tekstuele informatie en statische en dynamische numerieke informatie [Sommerville, 1992]
Het weergeven van deze verschillende soorten informatie is verschillend. Naast de voornamelijk tekstuele informatie uit het hoofdscherm, hebben we bij de rapportage meer te maken met het weergeven van numerieke informatie. Deze informatie heeft als doel om, duidelijk zichtbaar weergegeven, de roostermaker te ondersteunen. Het is dus belangrijk dat dat goed gebeurd. Het weergeven van die numerieke informatie kan op verschillende manieren en daarom kijken we naar de rapportage van de bouwbedrijven om te kiezen hoe we deze informatie op het scherm willen zetten.



Figuur 14 : Vraag en aanbodlijst PlanMate

In het document 'Rapportage bij de participerende bouwbedrijven' [van der Meer 1994]is een duidelijk overzicht gegeven van de rapportage zoals die nu bij de bouwbedrijven plaatsvindt. We kunnen er van uitgaan dat de manier waarop de informatie daarop wordt weergegeven een manier is die in de loop van de tijd het beste is gebleken. Het zou dus dom zijn om deze manier niet serieus te nemen en door de grote overeenkomst in rapportage tussen de verschillende bouwbedrijven neem ik aan dat de door de bouwbedrijven gebruikte manieren op dit moment de beste zijn.
Daar komt bij dat het roosterpersoneel niet aangepast moet worden aan PlanMate maar dat PlanMate geschreven moet worden voor de roostermakers. Probleem hierbij is wel dat er door de huidige software op rapportagegebied al een bepaalde verwachting bestaat over de kwaliteit van de rapportage waardoor het nodig zal zijn om in een aantal maanden de functionaliteit van commercieel populaire pakketten als Lotus 123 en andere spreadsheet applicaties te evenaren en dat geldt ook voor de kwaliteit van de printeruitvoer. Dat dit belangrijk is voor het uiteindelijke slagen van de applicatie is duidelijk, maar binnen deze afstudeerstage heeft het geen prioriteit.

De vraag en aanbod lijst
Bij bouwbedrijf Moes was de vraag naar een vraag en aanbod lijst aanwezig. Figuur M.3 staat in het document 'Rapportage bij de participerende bouwbedrijven' en is daar een voorbeeld van. Ik heb hiervan de lay-out zoveel mogelijk overgenomen en er ook voor gezorgd dat de roostermaker er later nog z'n eigen commentaar bij in kan voegen. Het is hier duidelijk dat de roostermaker dus behoefte heeft aan dynamische tekstuele weergave. Dit heeft denk ik te maken met de relatieve kleine lijst waarmee gewerkt wordt. Er is namelijk aan het einde van het roosterproces nog maar weinig vraag en aanbod over. Ik heb zo mijn twijfels of er nog goed te werken zou zijn met een vraag en aanbodlijst van enkele bladzijden, maar dat zal misschien in de toekomst blijken en dan, in samenwerking met de roostermakers, een aanpassing van het scherm vergen.
Er moet dus een duidelijke scheiding blijven tussen het genereren van de vraag en aanbodlijst en het scherm waarop deze lijst gepresenteerd moet worden.

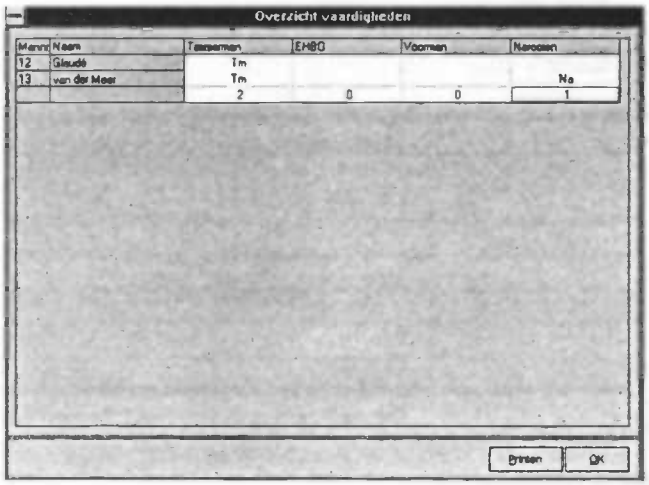


Figuur 15 : Personeelsbehoeftewindow PlanMate

De personeelsbehoefte
Capaciteitsbepaling is een belangrijk onderdeel voor de bouwbedrijven. Op deze manier kan men in de gaten houden dat er op de lange termijn genoeg werk is voor de werknemers en dat er genoeg werknemers zijn voor het werk. Het is dus zaak dat uitschieters in de personeelsbehoefte opgemerkt worden, zodat er tijdig actie ondernomen kan worden. Daarom is het

het handigst deze numerieke informatie grafisch en gerelateerd weer te geven. Dit wordt op dit moment ook gedaan bij al de participerende bouwbedrijven zoals in het rapportagedocument te zien is In figuur M.2, G3-5 en H.1-2.
Op dit scherm is goed te zien welke beroepen nodig zijn en of de totale behoefte aan werknemers niet te groot of te klein wordt. Deze vorm van rapportage heeft een hoop mogelijkheden voor uitbreidingen in zich omdat veel verschillende instellingen mogelijk zijn, bijvoorbeeld de behoefte per één of meer disciplines of de behoefte per één of meerdere projecten. Dit is echter allemaal toekomstmuziek en zal in latere versies waarschijnlijk gewenst zijn, maar is nu te bewerkelijk om binnen deze stage op te nemen.

De vaardighedenlijst
Jan Schriever, roostermaker bij Grootel, heeft grootse plannen met PlanMate en één daarvan was een uitgebreid personeelsbestand met een groot scala aan vaardigheden. Een belangrijke persoonlijke wens van hem was de mogelijkheid een uitdraai te kunnen krijgen van alle werknemers met al hun vaardigheden. Het was daarbij belangrijk dat in de te vormen matrix aan de cel te zien was welke vaardigheid de werknemer bezit, zodat de lijst duidelijk en overzichtelijk wordt.

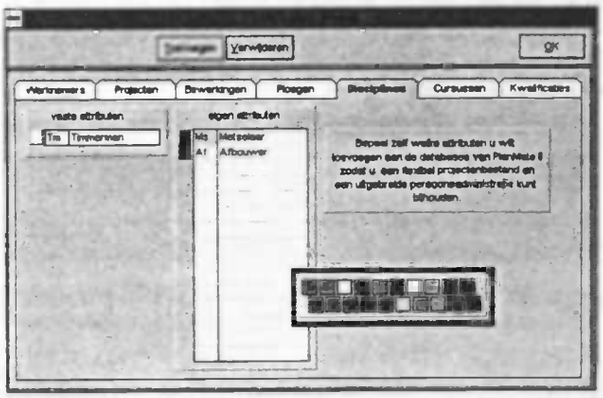


Figuur 16 : Vaardighedenlijst PlanMate

Deze wens is echter indirect bij Moes ook aanwezig gezien figuur M.5 in het rapportagedocument. Daarin wordt ook de vaardigheid van elke werknemer vermeld en omdat men daar de werknemers steeds meer in kaart wil gaan brengen wat hun vaardigheden betreft, zal deze lijst daar ook belangrijker gaan worden. Toekomstige uitbreidingen kunnen hier bijvoorbeeld zijn dat er in deze lijst veranderingen in de attribuutwaarden van de werknemers kunnen worden aangebracht en dat ook hier selectie- en sorteerroutines gebruikt kunnen worden.
Ook hier zijn uitbreidingen te bedenken. Zo is de lijst van vaardigheden van Moes anders opgezet dan die van Grootel en kan deze manier veel slimmer blijken te zijn. Door het kleine aantal bedrijven waarbij acquisitie is gepleegd, is dit niet goed in te schatten. Duidelijk is dat de manier van Moes minder papier en inkt kost. Het is echter onoverzichtelijker dan de wijze die Grootel hanteert.
De beste oplossing zal moeten blijken in de ronde van prototyping die plaats zal vinden.

Invoer van meerdere attributen
De personeelsroosteraar van Grootel is duidelijk bezig met het opzetten van een uitgebreide personeelsadministratie en hij wil PlanMate daarvoor gaan gebruiken. Andere roostermakers zijn echter veel minder bezig met het bijhouden van allerlei personeelsattributen en hebben meer oog voor de continuïteit binnen het bedrijf.
Deze verschillen in standpunten maken het aan de ene kant moeilijker een generiek systeem te bouwen waar alle roostermakers mee aan de slag kunnen. Aan de andere kant is het nuttig gezichtspunten die veel verschillen, te bestuderen. Zo kunnen we op alle onderdelen van PlanMate een zo uitgebreid mogelijk aanbod van functies genereren.
Waar we in ieder geval rekening mee moeten houden, is een groot verschil in attributen dat de ene roostermaker zal gebruiken ten opzichte van de andere roostermaker. Dit kunnen we opvangen door te zorgen dat roostermakers hun eigen attributen kunnen invoeren. Deze spelen vanzelfsprekend geen rol in de kennisbanken, maar bieden voor roostermakers, die veel attributen wensen, wel de mogelijkheid deze zelf te definiëren. Roostermakers die hier geen behoefte aan hebben zullen dit in een veel mindere mate doen. Een ander voordeel is dat het geen probleem is wanneer er een verschil in attributen, die roostermakers wensen, optreedt.
Het is echter een vereiste voor de evaluator en generator dat er een minimum aan attributen bestaat waaruit deze modules kunnen putten. Deze zijn dus niet te verwijderen ook al zou de roostermaker dit willen. Het moet wel mogelijk zijn deze attributen de waarde NULL te geven. De evaluator en generator moeten dan, wanneer ze deze gegevens wel nodig hebben, de informatie tijdens de sessie alsnog aan de gebruiker vragen. Het is dus belangrijk dat de roostermaker niet eigen attributen definiëert, die al in een andere vorm zijn opgenomen in PlanMate.

Dit zou betekenen dat de evaluator en generator langzaam en storend functioneren, omdat er steeds interactie met de gebruiker moet plaatsvinden over gegevens die eigenlijk al in het systeem opgenomen zijn.
Het invoerscherm bevat de vier gegevenstabellen die op het hoofdscherm te zien zijn en de vaardigheden discipline, cursus en kwalificatie. Bij deze vaardigheden is het noodzakelijk een unieke afkorting te geven omdat deze gebruikt worden bij het kort weergeven van de vaardigheden van de werknemer. Wanneer deze veel vaardigheden zou bezitten is het namelijk ondoenlijk deze allemaal overzichtelijk op het scherm te tonen. Ik ga er hierbij van uit dat roostermakers zelf al afkortingen gebruiken (zie rapportagedocument).
Wanneer dit niet het geval is zal de roostermaker volgens mij de afkortingen na een paar keer gebruiken begrijpen en herkennen.
Het valt nog op dat er kleuren moeten worden gedefiniëerd bij het invoeren van disciplines. Dit is gedaan, zodat op het hoofdscherm de stroken die verschillende disciplines vertegenwoordigen van elkaar te onderscheiden zijn. Dit werd vooral belangrijk gevonden door Aart Stolte van bouwbedrijf Hendriks.



Figuur 17 : Attributeninvoer in PlanMate

Overige zaken van belang

De kleuren
Bij het bespreken van de grafische mogelijkheden die in een in Visual Basic geprogrammeerd gebruikersinterface beschikbaar zijn, hebben we het nog niet gehad over het belang van kleuren. In de literatuur is men het er over eens dat kleuren functioneel zijn, maar dat een teveel aan kleuren en fout gekozen kleuren vermoeiend en verwarrend kunnen werken. Men spreekt hier van kleurvervuiling. Dit verschilt per applicatie. Het is handig te kijken naar de richtlijnen die Schneiderman [1986] opstelde. De belangrijkste daarvan zijn:

1. Informatie op het scherm mag niet alleen aan de kleur te herkennen zijn.
2. De gebruiker moet zelf de kleuren kunnen regelen.
3. Gebruik niet meer dan 7 kleuren in het hele interface.
4. Gebruik de kleuren op een consistente manier. Bijvoorbeeld rood betekent gevaar in het hele interface.
5. Gebruik kleuren om veranderingen in het systeem te laten zien. Kleurveranderingen vallen op en kunnen de aandacht op een bepaald object richten.

We kunnen zeggen dat we bij PlanMate niet teveel kleuren moeten gebruiken. De in het hoofdscherm gebruikte kleuren zijn zachte kleuren, zodat ze niet storend en vermoeiend zijn. De rapportagekleuren zijn puur functioneel en niet overdadig, maar er zal een instelling nodig zijn die het mogelijk maakt om met patronen in plaats van kleuren te werken, zodat ook kleurenblinde roostermakers onderscheid kunnen maken tussen verschillende disciplines.

De 'help'-ondersteuning
We zagen al eerder dat een ondersteuning door middel van allerhande hulpfuncties voor de meeste onervaren gebruikers onontbeerlijk is. Dit betekent een goede en constructieve begeleiding op het moment dat een gebruiker een fout maakt, maar ook een handleiding die de gebruiker kan gebruiken naast de applicatie.
Een instructieve tekst heeft zo z'n eigen opbouw en leent zich niet voor overmatige levendigheid. Wel kan er sprake zijn van een persoonlijke stijl maar het is belangrijk weinig vaktaal te gebruiken en de zinnen kort en begrijpelijk te houden. Waar we op moeten letten is dat we geen negatieve stijl gebruiken en ook geen lijdende vorm. Een gebiedende wijs klinkt misschien hard, maar is wel effectief.
Ik heb geprobeerd een handleiding te schrijven voor PlanMate II, die samen met het prototype getest moet gaan worden in de praktijk. Deze handleiding is bij mij te verkrijgen.

Conclusie

Wanneer we kijken naar de database en het elektronisch planbord wat we tot nu toe ontwikkeld hebben, lijkt PlanMate aardig de behoeften van de roostermakers te kunnen vervullen. Of dit ook werkelijk het geval is moet gaan blijken op het moment dat de roostermakers er mee zullen gaan werken. De kans is groot dat er tijdens die fase weer nieuwe zaken naar voren komen, die ondersteund moeten worden, en dat de efficiëntie vergroot zal moeten gaan worden. Daar is tot nu toe namelijk weinig rekening mee gehouden.

Hoofdstuk elf

Het programmeren van PlanMate II

Het gebruikersinterface in Visual Basic

Hoe besturen we een gebruikersinterface in Visual Basic
Door middel van de interfacebuilder die aan Visual Basic gekoppeld is, kunnen we snel en makkelijk een interface bouwen door een aantal objecten op het scherm te tekenen. Op deze objecten kunnen door handelingen van de gebruiker een aantal events plaatsvinden. Zo zit er onder een drukknop het event 'Click' dat geactiveerd wordt zodra de gebruiker op de knop duwt. In de procedure (in Visual Basic heet dit een Sub) die dan opgestart wordt, kan Visual Basic code geprogrammeerd worden.
Deze schermobjecten hebben allemaal een aantal properties, zoals bijvoorbeeld de kleuren, de grootte, de naam enzovoorts. Deze properties zijn bij de initialisatie in te stellen (bij het ontwerp van het interface), maar ook vanuit de programmacode. Dit geeft de mogelijkheid om bijvoorbeeld kleuren te veranderen of knoppen uit te schakelen. Ik zal een voorbeeld uit PlanMate geven ter verduidelijking.

Een voorbeeld uit PlanMate
Op het moment dat we beginnen, tekenen we het scherm en dus ook het roosterscherm. Dit gebeurt onder andere met een routine waarin de properties van de onderdelen van het scherm een waarde krijgen. Dit gaat op de volgende manier:

```
frmProjectBasis.picRooster.Left = 60
frmProjectBasis.picRooster.Top = 0
frmProjectBasis.picRooster.Height = Screen.Height
```

We zien dat een property altijd voorafgegaan wordt door de naam van het object waar het bij hoort. Wanneer we objecten in andere windows willen veranderen, moeten we de naam van het window weer voor de objectnaam vermelden. We zien dat dat hier **frmProjectBasis** betreft.

Datamanagement in Visual Basic

Welke mogelijkheden tot dataprocessing heeft Visual Basic?
Visual Basic heeft een zogenaamde Data Definition Language (DDL) waarmee datastructuren gedefinieërd kunnen worden. Deze bestaat uit functions, statements, methods en properties waarmee eigen database-tabellen, velden en indexen aangemaakt en bewerkt kunnen worden.
Dit is natuurlijk een eigenschap die grote mogelijkheden biedt voor een applicatie als PlanMate, die erg gericht is op het handig en overzichtelijk bewerken van gegevens.
Door middel van een speciale verzameling commando's kunnen we met databasegegevens omgaan alsof het gewone variabelen betreft.

Een praktijkvoorbeeld
In PlanMate kunnen we onze eigen attributen definiëren. Dit betekent dat we dus een veld moeten kunnen toevoegen in een tabel in een database. Hieronder volgt een deel van de procedure zoals dat gedaan is in PlanMate II. Deze zal ik vervolgens stap voor stap verduidelijken:

```
Sub VoegVeldToe (DatabaseNaam$, TabelNaam$, VeldWaarde$, Size%)
  Dim Db As Database
  Dim NewField As New Field
  Dim Fld As Fields
  Dim Boolean As Integer, t%

  Set Db = OpenDatabase(DatabaseNaam$, False, False, "")
  Set Fld = Db.TableDefs(TabelNaam$).Fields
  Boolean = True
  For t% = 0 To Fld.Count - 1
    If StrComp(Fld(t%).Name, VeldWaarde$, 1) = 0 Then
```

```
        Boolean = False
      End If
    Next
    If Boolean = True Then
      NewField.Name = VeldWaarde$
      NewField.Type = DB_TEXT
      NewField.Size = Size%
      Db.TableDefs(TabelNaam$).Fields.Append NewField
    End If
    Db.Close
End Sub
```

Databases openen in Visual Basic
Wanneer we een database willen openen zodat we er gegevens uit kunnen lezen, in kunnen veranderen en aan toe kunnen voegen, geven we de volgende commando's:

```
Dim Dbase As DataBase
Set Dbase = OpenDatabase ("Filename.MDB")
```

We hebben nu de databasevariabele **Dbase** gedefinieërd en door deze variabele kunnen we nu de inhoud van de database '**Filename.MDB**' bewerken.
We moeten nu nog een variabele definiëren die ons toegang geeft tot de inhoud van een bepaalde tabel. Dit kan door een 'Table'variabele, een 'Dynaset'variabele of een 'Snapshot' variabele. Elk van deze heeft zo z'n voordelen:

- Een Tablevariabele is alleen te definiëren door er een tabel aan te verbinden.
- Een Dynasetvariabele kan nadat hij is gekoppeld aan een tabel ook opnieuw geopend worden op zichzelf met een zekere selectiequery, waardoor we deelverzamelingen met gegevens kunnen maken van de oorspronkelijke tabel.
- Een Snapshotvariabele kan geopend worden van een tabel, een dynaset en zichzelf. Het grote verschil met een dynaset is dat een snapshot een copy bevat van de tabel en veranderingen in de snapshot niet verandert in de data in de database. Een dynaset bevat alleen de keys van de tabel, terwijl de snapshot alle data bevat en het werkstation dus zwaarder belast.

Het openen van dynasets
Ik heb voor de dynaset gekozen omdat daarbij het gebruik van SQL-queries het makkelijkst is en omdat een copy van de bestanden een te zware belasting voor het systeem zou worden wanneer er met uitgebreide bestanden gewerkt gaat worden. Zo'n dynaset open je als volgt:

```
Dim PersonenDynaset As Dynaset
Set PersonenDynaset = PlanMate.CreateDynaset ( "tabelnaam" )
```

of je opent hem op zichzelf:

```
Set PersonenDynaset = PersonenDynaset.CreateDynaset ( SQL )
```

Een tabel heeft een aantal objecten waar ook properties en methods bij horen. Een aantal van die objecten zijn: Fields (velden van de tabel), Index (de index van de tabel) en TableDefs (de tabelnamen van een database).
Deze TableDefs bevat de eigenschappen van de verschillende onderdelen van een tabel. Zo staat in ons voorbeeld in **Fld.Count** de hoeveelheid velden die de tabel "**personen**" bevat. Deze definities kunnen we veranderen en in ons voorbeeld doen we dat door middel van de **Append** method. We moeten wel eerst alle verbindingen met de database sluiten, omdat we anders niet bij de tabeldefinities kunnen. We geven ook aan dat we de veldencollectie van die bewuste tabelnaam gaan veranderen. Dit nieuwe veld heeft een aantal properties die gedefinieërd moeten worden. De naam, het type en in het geval van DB_TEXT (een tekstveld dus) de grootte. Deze definities moeten plaatsvinden voor de toevoeging.

Het toevoegen, veranderen en verwijderen van gegevens in dynasets
Op het moment dat er in het gebruikersinterface een object geselecteerd wordt en de bijbehorende stamkaart geopend wordt, kan er een verandering in de gegevens plaatsvinden. Dit gebeurt bijvoorbeeld voor personen in de volgende routine:

```
Sub PersoonNaarDynaset (Errors%)
  Dim t%
  Dim SelectieDynaset As Dynaset
```

```
  Set SelectieDynaset = PlanMate.CreateDynaset("SELECT Personen WHERE ManNummer LIKE '" & sManNummer & "' ")
  SelectieDynaset.Edit
  For t% = 0 To iGrid0Teller - 1
      SelectieDynaset(PersoonsAttributen(t%)) = Form3.Text1(t%).Text
  Next
  On Error GoTo DubbelNummer
  SelectieDynaset.Update
  SelectieDynaset.Close
End Sub
```

In deze routine is het mannummer van de geselecteerde persoon globaal bekend. De SQL query selecteert hier de andere werknemers uit de tabel, zodat we een verzameling van één overhouden.
iGrid0Teller is een globale variabele die bijhoudt hoeveel attributen de persoonstabel heeft. We hadden hier dus ook de **TableDefs("personen").Fields.Count** kunnen nemen zoals dat in een eerdere paragraaf gebeurde. We kijken vervolgens voor elk attribuut naar het type, zodat we niet integers in text-velden opslaan.
Met **SelectieDynaset.Update** geven we tenslotte aan dat de veranderingen die we in de dynaset hebben aangebracht, ook in de werkelijke database veranderd moeten worden.
Dit zijn de belangrijkste commando's uit de DDL van Visual Basic.

Sorteren en selecteren in Visual Basic
Wanneer we het sorteren in PlanMate willen bekijken kunnen we kort zijn. Dankzij het commando **SelectieDynaset.Sort = "[Name]"** wordt een dynaset bijvoorbeeld gesorteerd op de waardes van het veld Name. Het is natuurlijk niet duidelijk of deze sorteeroptie het snelste sorteeralgoritme bevat dat op het moment beschikbaar is, maar het valt te betwijfelen of een zelf geïmplementeerde routine het sorteren sneller zou kunnen dan de ingebouwde functies van Visual Basic.
Het selecteren heeft wat meer voeten in aarde. Dankzij de beschikbare SQL-driver kunnen er in Visual Basic met SQL-queries selecties uitgevoerd worden op de databasebestanden. Deze commando's zijn echter erg traag wanneer er een lange query ingevoerd wordt. Daar komt bij dat wanneer we op vaardigheden willen selecteren, we met 7 tabellen uit de database te maken hebben en we dus een lange, ingewikkelde query krijgen. Dit was dus wel een mogelijkheid, maar het bleek veel sneller en simpeler om op het attribuut 'vaardigheid' van personen te selecteren. We stellen eenmaal de selectiestring vast en selecteren vervolgens door deze string te vergelijken met het attribuut 'vaardigheid' van de personen.
Eén van de wetten in de software engineering is 'wanneer je software moet ontwikkelen die tegelijkertijd ook efficiënt is, doe het dan maar niet!'. Toch heb ik voor deze oplossing gekozen omdat het een centrale bewerking op het gegevensbestand is die niet storend langzaam moet zijn.

Rapporteren in Visual Basic

We merkten al eerder op dat vierde generatie talen goede rapportagemogelijkheden met zich meedragen. Visual Basic heeft dit ook doordat zogenaamde VBX-files beschikbaar zijn die grafieken of tekst op een verzorgde manier snel en goed kunnen weergeven. Het aansturen van deze objecten is in de bijgaande handleidingen goed beschreven, dus dat zal ik hier achterwege laten. Wel zal ik aangeven hoe ik de te rapporteren gegevens verkrijg in PlanMate II.

De berekeningen
Ik zal de berekening van de vraag op de vraag en aanbodlijst als voorbeeld even verduidelijken. Ik zal kort de gevolgde procedure uitleggen. Het moet dan niet ingewikkeld zijn deze in de programmatekst terug te vinden. Deze is opgenomen in bijlage C.

We beginnen met de aanroep **call VraagLijst** en belanden dan dus in de onderstaande procedure:

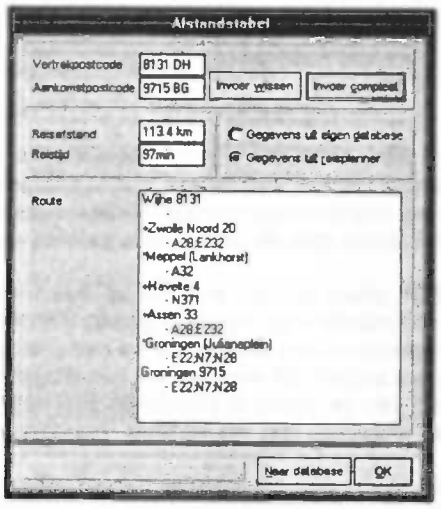
- Bepaal eerst alle bewerkingen die gedeeltelijk of helemaal in de door de gebruiker gedefinieerde periode vallen.
- Wanneer er records geselecteerd zijn door middel van de SQL query, kunnen we naar de eerste wijzen en een header voor de uiteindelijke lijst aanmaken. Wanneer er geen records zijn hoeft dit niet
- We gaan nu alle records in de dynaset af en bepalen eerst een aantal van hun attribuutwaarden, zodat deze op de lijst geplaatst kunnen worden.
- Daarna bepalen we de periode die we moeten bekijken: De tijdsgrenzen van de bewerking en/of die van de periode waarover de lijst gemaakt moet worden. (we hoeven niet buiten die periode te zoeken, wanneer een bewerking daar nog wel gepland is)
- We roepen nu de procedure **VraagBepalen** aan.
- Hierin bepalen we welke personen ingeroosterd zijn op de bewerking.
- Van al deze personen bekijken we in de tabel 'procenten', hoeveel procent ze ingeroosterd zijn op de bewerking.

- Wanneer we dit van alle werknemers opgeteld hebben, trekken we dit af van de totale vraag, en bepalen zo de behoefte.
- Wanneer deze behoefte groter dan 0 is gaan de nodige gegevens naar een tekststring. **(MaakVraagLijst)**

Op deze manier haal ik dus gegevens uit de databases en verwerk ze, zodat ze in een rapportage window getoond kunnen worden.

De afstandstabel

Omdat een belangrijke behoeftevraag het minimaliseren van de reiskosten is, is het nodig om van elke werknemer te weten wat zijn reisafstand tot andere plaatsen is. Dit kunnen werken zijn, maar ook woonplaatsen van andere werknemers. Daarom is er met behulp van AND software de mogelijkheid geboden een afstandentabel te koppelen aan PlanMate. Veel roostermakers gebruiken nu al een reisplanner om afstanden te kunnen bepalen en het is dus een goede ondersteuning wanneer er een koppeling mogelijk is die voorkomt dat de roostermakers zelf al deze gegevens in moet voeren. Nadeel is wel dat deze koppeling niet erg snel is. Dit gaat een rol spelen wanneer we bij het genereren van roosters veel afstanden nodig hebben.



Figuur 18 : Afstandstabel PlanMate

De afstandstabel in het gebruikersinterface

PlanMate moet een systeem worden dat gekoppeld aan andere systemen moet kunnen functioneren. Het moet echter ook mogelijk zijn dat het programma op zichzelf draait, en dat houdt in dat gegevens in- en uitgevoerd moeten kunnen worden. Daarom heb ik gekozen voor een invoerwindow voor de afstandstabel. Wanneer er geen koppeling is met de afstandstabel kan de roostermaker hier zijn eigen waarden invoeren, of de waarden die hij of zij al had ingevoerd opvragen. Op het moment dat er wel een koppeling is, kan de roostermaker de waarden die hieruit gehaald worden veranderen wanneer er in overleg met werknemers bijvoorbeeld in verband met een omleiding een andere hoeveelheid kilometers is afgesproken.

De gegevens uit deze zelfgemaakte tabel zullen dus gebruikt gaan worden in de evaluator en generator.

In de versie van PlanMate zoals die nu bestaat is de mogelijkheid opgenomen de route te bepalen tussen verschillende plaatsen. Hier is het ook mogelijk dat er gebruik wordt gemaakt van een procentuele inroostering per persoon per tijdsperiode. Dit was duidelijk een wens vanuit KPD nadat een geïnteresseerde klant aangaf dit graag in PlanMate te zien.

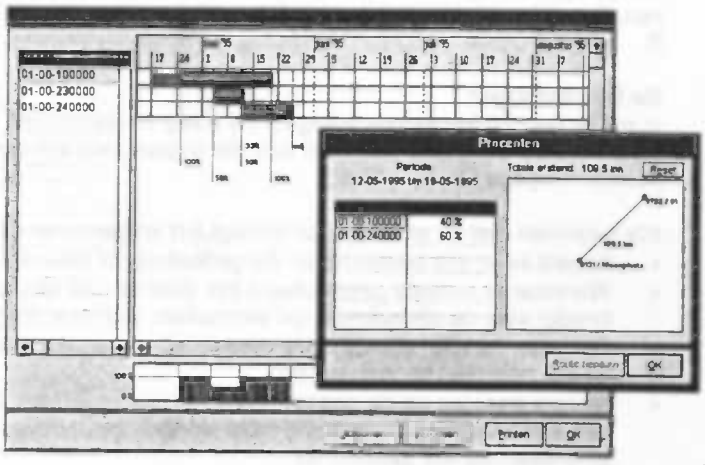
Vergeleken met de eerste versie van PlanMate moeten we dus niet een groep werknemers met zo weinig mogelijk kilometers op de werkvloer krijgen, maar een werknemer met zo weinig mogelijk reisafstand op een groep werken. Dit heeft een geheel andere insteek van het planningsprobleem. In het eerste geval was namelijk de ploegindeling het Traveling Salesman Problem. Bij het procentueel inroosteren van personeel, wordt echter de route van werk naar werk veel belangrijker bij het minimaliseren van de reiskosten.

Hoe roepen we de afstandstabel aan

Door AND Software is een DLL (Dynamic Link Library) geleverd die vanuit Visual Basic aan te roepen is. Het volgende gedeelte uit een routine haalt gegevens uit deze DLL [AND 1994]:

```
{ declaraties variabelen }

  fromnode = TDNODENR(Postlaag)
  tonode = TDNODENR(Posthoog)
      retval = TDGETTDC(Postlaag, Posthoog,
ttime, dist, cost)
    retval = TD_GETPATHORDERED(fromnode,
tonode, nodes(0), roads(0), pathlen + 1)
  If (retval <> 0) Then
    For counter = 0 To (retval - 1) Step 1
        retval = TDNODENAME(nodes(counter),
nodename)
        retval = TDROADNAME(roads(counter),
```



Figuur 19 : Personeelsroostering procentueel met route in PlanMate 0.3

```
roadname)
    Next
  End If
```

De functie-aanroepen die beginnen met TD zijn de DLL-functions. Zij moeten in het declaratieve gedeelte van een module als volgt vermeld worden:

```
Declare Function TDGETTDC% Lib "td.dll" (ByVal fromname$, ByVal toname$, ttime%, dist%, cost&)
```

Dit gebeurt om aan te geven dat deze functie in td.dll gezocht moet worden.
Op deze manier is het dus verbluffend simpel om de C-routines uit de AND DLL aan te roepen.

Hoofdstuk twaalf

Het rekenen in het roosteren

In het begin van dit document merkte ik al op dat een kennissysteem bestaat uit een beredenerend systeem in de vorm van kennisbanken en een berekenend gedeelte in de vorm van een computer. Het is dus duidelijk dat er ook rekenwerk plaats zal vinden achter het gebruikersinterface van PlanMate. De berekeningen binnen PlanMate hebben vooral te maken met personeelskosten en reiskosten. De personeelskosten zijn het laagst, wanneer de ploegcontinuïteit het best is. De reiskosten zijn daarentegen moeilijker te minimaliseren, omdat we hier een algoritme nodig hebben dat kortste routes moet kunnen berekenen (een traveling salesman problem), maar daarvoor moet een ploegindeling berekend worden. Dit laatste probleem wil ik proberen op te lossen, omdat een goede ploegindeling eigenlijk een TSP-algoritme overbodig maakt.

De algoritmes in de kennisbanken

De implementatie in Nexpert, waar we bepalen welke werknemers op welke bewerkingen terecht komen, is in feite een algoritme. Een voorbeeld van zo'n algoritme is het bepalen van de dichtst bij een bewerking wonende ploeg. Voordat we dit algoritme uit kunnen werken in een stappenplan en een verzameling regels, moeten we een aantal aannames maken:

1. De bevolkingsdichtheid in Nederland is zodanig dat het aannemelijk is dat veel ploegen binnen een straal van 5 kilometer bij elkaar vandaan wonen. We hebben het dan over dorpen en steden.
2. Wanneer twee postcodes meer dan 20 kilometer van elkaar liggen, spreken we niet meer van een geschikte afstand voor twee ploegleden.
3. Een ploeg heeft tussen de 4 en 9 leden. Dit wordt bepaald aan de hand van de beschikbare vervoersmiddelen. Een busje kan 9 mensen vervoeren, en privé-auto's vervoeren er 4.
4. We stellen de hoogste prioriteit aan de minimale reiskosten.

Deze aannames zijn niet helemaal uit de lucht gegrepen, maar naar aanleiding van de gesprekken met de roostermakers en met het gezonde verstand aangenomen. Binnen het algoritme zijn het echter factoren die gevarieerd kunnen worden, waardoor we altijd andere waarden kunnen gebruiken.

Hoe kiezen we de dichtsbijzijnde ploeg?

In het kort komt dit probleem neer op de volgende stappen:

1. Kies de werknemers die voldoen aan de harde randvoorwaarden met betrekking tot de disciplines en beschikbaarheid.
2. Kies de dichtst bij elkaar wonende X werknemers.
3. Ken de ploegen 100% toe aan de werken waar de discipline van de ploeg gewenst is en die op dat moment actief zijn, zodanig dat de vraag niet overschreden wordt.
4. Kies bij een gelijkwaardige keuze voor ploegcontinuïteit.
5. Herhaal deze stappen voor de andere disciplines.
6. Verdeel de overgebleven werknemers zodanig dat er EHBO'ers en voormannen op de werken aanwezig zijn en plaatsen in auto's opgevuld worden.

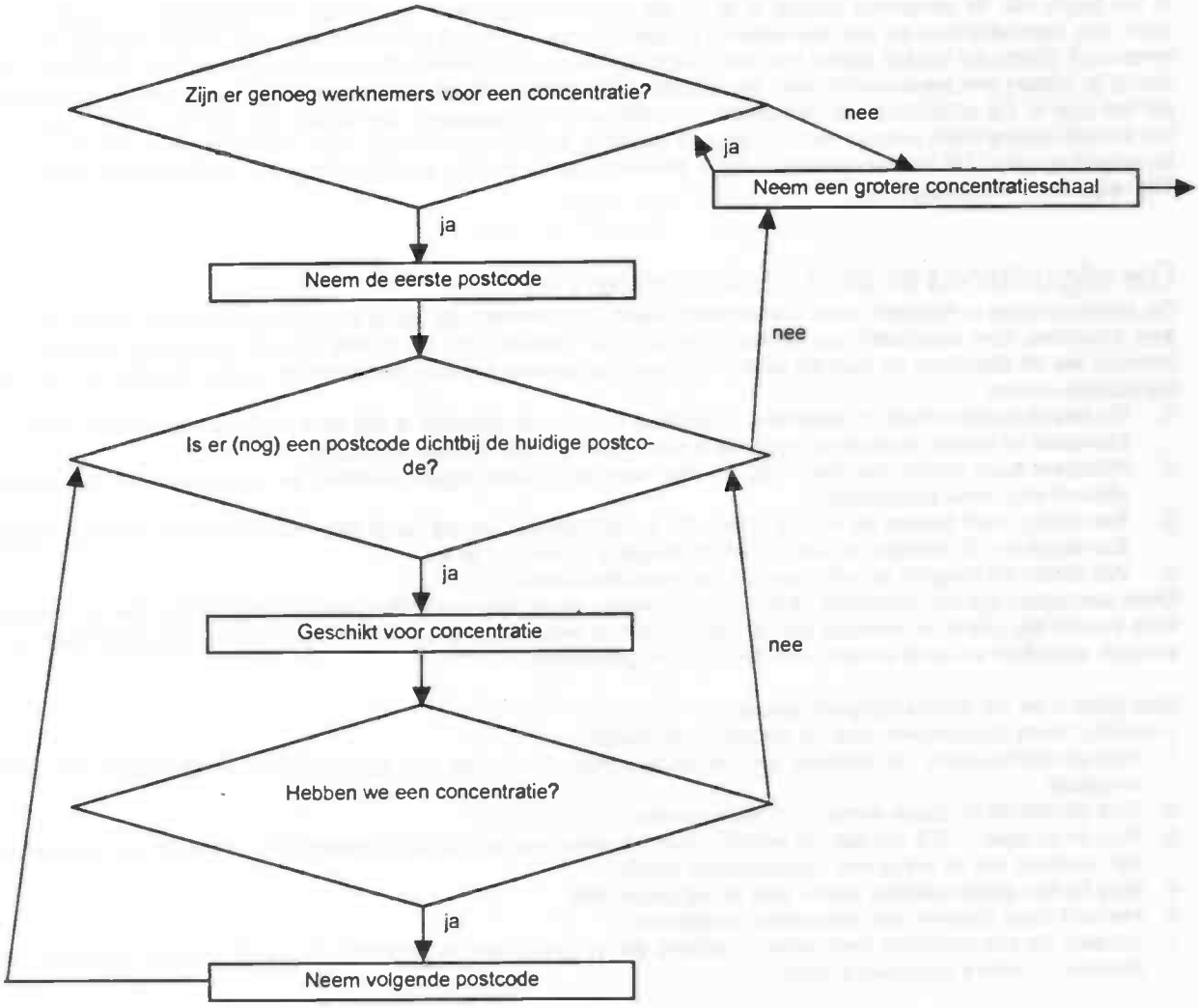
De randvoorwaarden en doelfuncties in het algoritme

We zien in de stappen die we in de vorige paragraaf namen, dat wanneer we ons strikt aan de daar bepaalde selectiecriteria houden, er al aan een groot gedeelte van de randvoorwaarden voldaan wordt. We leggen hier dus een soort filter over de database. Hiermee selecteren we voor de koppeling van werknemers aan werken al een aantal werknemers uit, zodat onze probleemruimte verkleind en het genereren versneld wordt.

Op bepaalde keuzemomenten kunnen we naar onze doelfuncties kijken om zo de beste keuze te maken. Wanneer we bijvoorbeeld meerdere kandidaten voor een ploeg hebben, kunnen we de ploeg samenstellen, die in vorige periodes ook al samenwerkte. Ook bij het toewijzen van ploegen aan werken kunnen we in de gaten houden welke ploegen al op dat werk gewerkt hebben. Dit scheelt inwerkperiodes en hiermee gebruik je ook weer een stuk kennis van de gebruiker, door naar roosters te kijken die hij goedgekeurd heeft.

Een stappenplan

Een stappenplan zou er als volgt uit kunnen zien:



Figuur 20 : Stappenplan ploegsamenstelling

Dit stappenplan is uiteraard niet volledig, maar schetst een goed beeld van de basis van het algoritme. De volgende stap is het stappenplan in heldere en duidelijke kennisregels om te zetten.

De kennisregels

De belangrijkste kennisregels die uit dit stappenplan gehaald kunnen worden, zijn hieronder weergegeven. Het zijn geen regels die zo in Nexpert zouden kunnen worden ingevoerd, maar ze bezitten wel de goede structuur zodat een vertaalslag niet al te veel problemen moet opleveren. We kunnen dit algoritme uitbreiden door te proberen de overblijvers per discipline nog in ploegen in te delen. Hier is echter te weinig acquisitie naar gedaan om het goed uit te voeren.

1. Maak een lijst van alle postcodes van beschikbare werknemers met dezelfde discipline
2. Zet de afstand waarbinnen de postcodes moeten liggen op 5 kilometer

Bepaal_ploegen

ALS de afstand waarbinnen de ploegleden wonen kleiner is dan 20 kilometer
DAN kunnen we een concentratie vinden
EN nemen we de eerste postcode uit de lijst
EN proberen we een concentratie te vinden

ALS we een postcode hebben
EN deze heeft binnen de bepaalde afstand genoeg andere postcodes in de buurt
DAN hebben we een concentratie
EN slaan we deze op als concentratie
EN halen we ze uit de lijst van beschikbare postcodes

ALS een postcode minder dan de bepaalde afstand verwijderd is van het bepaalde punt
DAN is de postcode geschikt voor een concentratie rond het punt

ALS er meer dan 2 en minder dan 9 postcodes geschikt zijn voor een concentratie rond het punt
DAN hebben we genoeg andere postcodes binnen de concentratie

ALS we geen concentratie hebben
EN er zijn nog postcodes beschikbaar op de lijst
DAN zijn er mogelijk nog andere concentraties
EN nemen we de volgende postcode uit de lijst
EN bepalen we de postcodes die binnen de bepaalde afstand daarvandaan liggen

ALS we geen concentratie hebben
EN we hebben de lijst helemaal afgewerkt
DAN zijn er geen concentraties meer mogelijk met deze afstand
EN verhogen we de afstand met 5 kilometer
EN proberen we een concentratie te vinden

Binnen de bovenstaande regels zijn er een aantal die erg bewerkelijk zijn en veel tijd kosten. De regel die kijkt of een postcode niet te ver verwijderd is van het bepaalde punt, moet bijvoorbeeld wel alle afstanden hebben tussen het bepaalde punt en de andere postcodes totdat een dichtbijgelegen postcode gevonden is. Wat we hier kunnen doen is gebruik maken van de postcode-indeling van KPN, en daar eerst op sorteren vanaf de bepaalde postcode. Dit scheelt waarschijnlijk een hoop zoekwerk naar dichtbijgelegen postcodes, omdat deze indeling streekgebonden is.
Ik zal in deel vier aangeven hoe deze regels (via Nexpert) in Visual Basic geïmplementeerd zouden kunnen worden.

Deel 4

Een aanzet tot de
ontwikkeling van
een vertaler voor Nexpert-uitvoer

Inleiding

Door het gebruik van Nexpert bij het realiseren van kennissystemen is het noodzakelijk een hardware device van Neuron Data mee te leveren. Hieraan zijn echter een aantal nadelen verbonden waardoor het gebruik van de shell commercieel gezien een stuk minder interessant wordt.

1. Een kennissysteem wordt een stuk duurder en dus minder interessant voor het bedrijfsleven.
2. Het aanroepen en laten draaien van Nexpert is traag.

Naast de voordelen die het commercieel gezien zou hebben, is het natuurlijk wetenschappelijk gezien ook interessant te bekijken hoe kennisbanken er in een hogere programmeertaal uitzien.

Omdat de uitvoer van Nexpert in een 'tekst'-vorm gegeven kan worden, moet het mogelijk zijn deze uitvoer te scannen en via een parser om te zetten naar een hogere programmeertaal. Wat dan uiteraard ontbreekt is de objectgeörienteerdheid van Nexpert en de ingebouwde redeneermechanismen.

De hierboven genoemde redeneermechanismen en objectgeörienteerdheid moet dus in de bewuste programmeertaal geïmplementeerd worden. De gekozen taal is Visual Basic, omdat KPD geïnteresseerd is in kennissystemen en programmeert in Visual Basic. In de bouw wordt voornamelijk backward en forward chaining toegepast, en daarom zal op het ontwikkelen van dit redeneermechanisme de nadruk liggen.
Het is misschien een idee ooit deze vertaalstappen in een snellere (en objectgeörienteerde) programmeertaal te herhalen zodat nog meer efficiëntie mogelijk is.

Hoofdstuk dertien

De uitvoer van Nexpert

Wat is de uitvoer van Nexpert?

Wat zit er wel in de Nexpert file?

De uitvoerfiles van Nexpert bevatten de door de gebruiker ingetypte informatie, met daarbij vermeld of het bijvoorbeeld om een CLASS of PROPERTY gaat, bv.:

```
(@PROPERTY=        postcode        @TYPE=String;)
(@PROPERTY=        functie         @TYPE=String;)

(@CLASS= werknemers
        (@PROPERTIES=
                postcode
                functie
                .
                .
        )
)

(@OBJECT= werknemer_1
        (@CLASS= werknemers
        )
)
```

enz.

We zien dat de uitvoer beperkt blijft tot een klein aantal symbolen en dat alle onderdelen op dezelfde manier door haakjes zijn omgeven. Daarbij is er een vaste opbouw in de weergave van de verschillende onderdelen van Nexpert:

- VERSION: Hier wordt de versie van Nexpert gegeven waarin de kennisbank is gemaakt.
- PROPERTY: Dit zijn de properties en hun type, van de classes die later gedefinieerd worden.
- CLASS: Hier worden de in Nexpert gedefinieerde classes weergegeven, met hun attributen.
- OBJECT: De objecten uit Nexpert met hun waardes.
- SLOT: De gedefinieerde slots met hun commando's
- RULE: De ingevoerde rules met hun nummer, de lefthandside, de naam en de righthandside.
- GLOBALS: De instellingen die in de Nexpertshell waren gemaakt. Behalve de waarde van SUGLIST (de regel waarvanuit begonnen moet worden) is dit niet van belang.

Deze strakke indeling in de uitvoerfile en de opbouw van elk onderdeel, maken het een stuk makkelijker een parser te schrijven. Aan het eerste woord kan namelijk gezien worden met wat voor een onderdeel we te maken hebben, waarna we naar een procedure kunnen springen die de specifieke invoer van dat onderdeel kan herkennen.

Wat zit er niet in de uitvoerfile

In de Nexpertfile zit alleen de door de gebruiker ingevoerde tekst met daarvoor vermeld wat de betekenis van de tekst, binnen Nexpert, is. Wat Nexpert vervolgens met deze tekst doet, is natuurlijk niet opgenomen in de uitvoerfile. Deze redeneermechanismen en de mogelijkheden die objectgeörienteerdheid met zich meebrengen zijn dus dingen die we zelf nog in Visual Basic moeten implementeren. Verder beschikt Nexpert over een groot aantal functies die door middel van het 'execute'-statement gebruikt kunnen worden. In Visual Basic moeten hier aparte procedures voor geschreven worden. Gelukkig gebruikt de gemiddelde Nexpertgebruiker niet veel van deze functies en zal het in eerste instantie het belangrijkst zijn de meest voorkomende functies te implementeren.

Hoe lezen we de Nexpertfile?

Om te kijken hoe we de Nexpertfile het best kunnen vertalen, moeten we de inhoud van elk onderdeel wat beter bestuderen. Op de eerste plaats is dan belangrijk hoe de uiteindelijke vertaling in Visual Basic er uit moet gaan zien. Groot voordeel is uiteraard dat we niet bang hoeven te zijn dat we foute invoer krijgen, zodat er bijvoorbeeld niet bijgehouden hoeft te worden of bepaalde properties wel tot een class behoren.

Veel onderdelen van Nexpert blijken makkelijk om te zetten in Visual Basic, wat betekent dat voor elk ingelezen rijtje tokens er direct naar de Visual Basic module geschreven kan worden. Het is echter niet bekend of dit voor elke functie en elk commando zo is en dan zou het erg onhandig zijn wanneer we daar achter zouden komen op het moment dat we de vertaler willen uitbreiden. Dit zou betekenen dat we het grootste gedeelte moeten herschrijven of met ingewikkelde constructies aan het werk moeten. Het veiligste lijkt daarom een tussenstap tussen de Nexpertfile en de Visual Basicfile waarin we de informatie uit Nexpert opslaan in een database zodat we daarop bewerkingen kunnen uitvoeren. Dit geeft met het oog op conversies naar andere hogere programmeertalen ook een grotere flexibiliteit, omdat dan dezelfde scanner gebruikt kan worden.

Conclusie

De uitvoer uit Nexpert lijkt zodanig gestructureerd dat het scannen en parsen weinig problemen op zal leveren. Wanneer we de informatie vanuit Nexpert naar Visual Basic willen vertalen, moeten we weten wat de verschillende onderdelen inhouden en hoe die het best geïmplementeerd kunnen worden in Visual Basic.

Hoofdstuk veertien

De implementatie in VB

Hoe brengen we de objecten in Visual Basic?

Hoe implementeren we de objecten uit Nexpert?

De objecten uit Nexpert hebben allemaal een bepaald type. Deze types kunnen we uit de file halen en onthouden.

```
(@OBJECT=       bepaal_ploegen
        (@PROPERTIES=
                Value     @TYPE=Boolean;
        )
)
```

Het is het best om er van uit te gaan dat we van elk object een functie maken met het variabelentype van het object in Nexpert, omdat deze objecten meestal een SLOT of een RULE zijn.

```
Function bepaal_ploegen ( ) As Boolean

End Function
```

Bij het implementeren van de rules en slots blijven er automatisch enkele objecten over die als propertie slechts één waarde hebben. Deze kunnen we dan gewoon als variabele implementeren.

Hoe implementeren we de regelstructuur in Visual Basic?

De hypotheses in Nexpert zijn in feite gewoon objecten met een booleanwaarde, die weer afhankelijk zijn van andere objecten en variabelen. Kijken we naar de Functions in Visual Basic (en tegenwoordig bijna iedere programmeertaal) dan is zo'n function een variabele, en kunnen binnen die functie andere variabelen (waaronder ook weer functions) bewerkt worden. Dit is een structuur die veel overeenkomsten vertoond met die van Nexpert.
Als we kijken naar de regels die we geproduceerd hebben, moet er eerst een selectie gemaakt worden die bepaalt dat alle geselecteerden dezelfde discipline bezitten. Dit wordt dan als volgt omgezet:

```
(@RULE=         R1
        (@LHS=
                (=   (werknemer_1.functie) (Timmerman))
        )
        (@HYPO=  bepaal_timmerman)
        (@RHS=
                (Show ("b:\PM_II\Text.txt")      (@KEEP=FALSE;@WAIT=TRUE;@RECT=0,0,600,100;))
        )
)
```

Wanneer we dit in Visual Basic willen implementeren wordt dit:

```
Type werknemers
        postcode As String
        functie As String
End Type
Dim werknemer_1 As werknemers
Function bepaal_timmerman ( ) As Boolean
        If werknemer_1.functie = "timmerman" Then
                bepaal_timmerman = False
                Exit Function
        End If
        bepaal_timmerman = True
End Function
```

Met als aanroep:

```
If bepaal_timmerman Then
        Call show ("b:\PM_II\Text.txt", FALSE, TRUE, 0, 0, 600, 100)
End if
```

N.B. In Visual Basic kennen we geen Booleans, maar dit type is makkelijk te implementeren.

Hoe implementeren we slots?

We moeten de events, die bij de objecten horen, ergens kwijt. In Nexpert zijn dit de zogenaamde SLOTS. De makkelijkste manier lijkt voor elke SLOT een **Sub** te maken, en elke aanroep naar een variabele A te vervangen door de **Sub** slot_A (A), wanneer de waarde nog niet bekend is of wanneer we niet met een toekenning te maken hebben. Deze procedure voert het SLOT uit en geeft de nieuwe waarde van A, bv.:

```
(@SLOT=         werknemer_1.functie
        @PROMPT="Wat is de functie van  @(tijdelijke_naam) ?";
        (@SOURCES=
                (Do     (werknemer_1.naam) (tijdelijke_naam))
        )
)
```

Wordt bijvoorbeeld in Visual Basic:

```
Sub slot_we_fu (werknemer As werknemers)
        Dim tijdelijke_naam As String

        Call sources (werknemer.naam, tijdelijke_naam)
        Call prompt ("Wat is de functie van " & tijdelijke_naam & "?", werknemer.functie)
End Sub
```

Met als aanroep in de programmatekst:

```
Call slot_we_fu (werknemer_1)
```

We zien hier dat we 'sources' voor 'prompt' moeten plaatsen, en dat we dus eerst naar het binnenste koppel haakjes moeten kijken bij een SLOT met een @ in de 'prompt'-string om de variabele die in de string gebruikt wordt, toe te kennen.

Het implementeren van de ingebouwde functies

Door middel van het 'execute'-commando kan in Nexpert een extra functie aangeroepen worden, zoals bijvoorbeeld "WriteTo". Het is het best deze functies als sub te implementeren en het 'execute ("WriteTo")'-commando bijvoorbeeld te vervangen door 'call WriteTo (Parameters)'. Er zijn een heleboel parameters nodig bij sommige functies, maar de ervaring leert dat daar maar weinig gebruik van gemaakt wordt. Het is dus ook niet nodig deze allemaal op te nemen. Wanneer in latere versies van de vertaler deze wel nodig zijn, is de parameter makkelijk toe te voegen.

Waar halen we de SUGGEST-rule vandaan?

Aan het einde van de Nexpertfile staan een aantal globale variabelen die de instellingen van de Nexpertvariabelen bevatten. In eerste instantie is alleen de rule die geSUGGEST wordt van belang zodat we weten waar we moeten beginnen met het valideren van de rules. Andere variabelen zijn aan hun naam niet zodanig te herkennen als belangrijke instellingen in de Nexpert-shell. Daarom zal ik daar pas aandacht aan besteden wanneer uiteindelijk blijkt dat deze wel nodig zijn.
Bij onze regels moeten we dus als SUGGEST waarde een functie aanroepen die BEPAAL_PLOEGEN moet valideren en waarin een aantal variabelen geïnitialiseerd worden.

Conclusie

De conversie van objecten uit Nexpert lijkt niet al te ingewikkeld te zijn. Door een voorbeeld te geven van een conversie van elk onderdeel weten we hoe we de vertaling moeten aanpakken en waar we op moeten letten. Op die manier lijkt de uitvoer van Nexpert goed om te zetten naar Visual Basic, maar of dit in alle praktijkgevallen ook zo zal

zijn moeten we afwachten. Duidelijk is dat wanneer we de Nexpert-shell kunnen nabootsen, het inlezen van de gegevens geen problemen hoeft te geven.

Hoofdstuk vijftien

De objecten en het redeneren

Objectgeörienteerdheid

Jacobson [1993] heeft in zijn OOSE methode beschreven hoe objectgeörienteerdheid volgens hem geïmplementeerd kan worden in een niet objectgeörienteerde programmeer-omgeving.

Het impedance probleem

In de eerste plaats zitten we met het *impedance problem*. Dit houdt in het kort in dat we de datatypes uit de objectgeörienteerde omgeving om moeten zetten naar de wat simpeler types in Visual Basic.
Het is het handigst een database aan te maken voor de objectgegevens uit Nexpert. Voor elke CLASS maken we een tabel waarin elke PROPERTY van die CLASS een veld wordt. We voegen daarbij een veld 'naam' toe, waarin bij het instantiëren van de verschillende OBJECTEN de unieke naam van het object komt te staan. Dit is dus de primary key. Relaties tussen objecten worden nieuwe tabellen, met de primaire keys van beide objecten.

De inheritance

Een tweede probleem is de inheritance, ofwel overerving. Omdat we werken met een geconverteerd programma waarin het niet mogelijk hoeft te zijn veranderingen aan te brengen in de basis structuur, kiezen we voor een kopiëren van de PROPERTIES van de parent CLASS naar de child CLASSES. Dit is voornamelijk voor de snelheid het meest gunstig. Wel is er het gevaar dat de omvang van de database een stuk groter wordt.

Events

Als laatste probleem is er het afhandelen van de events. Dat werd in hoofdstuk veertien al opgelost door deze te vervangen door een procedure, waardoor we de event-aanroep kunnen vervangen door de procedure-aanroep.

Redeneermechanismen

Dit is een wat ingewikkelder onderdeel, omdat er verschillende redeneermechanismen in Nexpert zijn. In deel 2 werd al uitgelegd wat we onder redeneermechanismen verstaan en welke we meestal tegenkomen. In de bouwwereld gebruikt men voornamelijk backward en forward chaining, dus op die twee mechanismen zal ik me richten.
Nexpert gebruikt een boomstructuur, die we goed kunnen zien wanneer we de browser in de ontwikkelshell gebruiken. Deze structuur hebben we in deel 3 al geïmplementeerd in Visual Basic.

Backward chaining in Visual Basic

Wanneer we backward chaining toe willen passen gaan we dus een bepaalde hypothese 'true' proberen te maken door de voorwaarden aan de lefthandside te toetsen. Globaal gezien gebeurt dit ook in Visual Basic zodra we de eerste function aanroepen, deze zal namelijk weer andere functies aanroepen totdat de hele function is doorlopen. Wanneer een waarde 'false' optreedt zal overgeschakeld worden naar forward chaining, om te kijken of er andere hypotheses wel true gemaakt kunnen worden.

Forward chaining in Visual Basic

Bij forward chaining hebben we een probleem. Wanneer een function 'true' blijkt te zijn, is het onmogelijk in andere functions te kijken of daar een functioncall voorkomt naar die bewuste function. Voorlopig lijkt het mij daarom het slimst iedere function die bij het converteren gemaakt wordt (dus iedere hypothese en elk slot) een nummer te geven, en voor iedere function een lijst aan te maken waarin we bijhouden in welke andere functions hij voorkomt. Het is dan nodig een procedure te schrijven die als volgt iedere function kan aanroepen:

```
Sub call_function (FunctieLijst As Array)

        For  t = 0 to Upperbound(FunctieLijst)
                Select Case  FunctieLijst(t)
                Case 1
                        dummy = bepaal_timmerman
                Case 2
                    ...
                    .
```

```
            End Select
        Next
End Sub
```

Op het moment dat zo'n function dan 'true' blijkt te worden, wordt deze function vooraan de lijst gezet van functions die gevalideerd moeten worden. Op deze manier wordt doorgegaan tot de beginfunction eventueel TRUE is.

Conclusie

Het blijkt dat het mogelijk is om de redeneermechanismen uit Nexpert in grote lijnen om te zetten naar programmastructuren in Visual Basic. Omdat de opbouw van Visual Basic geen grote verschillen vertoond met veel andere hogere programmeertalen, is de mogelijkheid om de vertaling naar die talen te laten plaatsvinden nog altijd aanwezig. Opgemerkt moet wel worden dat ik wel alleen heb gekeken naar de basis van Nexpert en niet naar de uitgebreide hoeveelheid functies die aanwezig zijn. Dit is echter meer een kwestie van het nabouwen van de functies dan het vertalen van de uitvoer.

Hoofdstuk zestien

Het algoritme

Nu we weten dat het in principe mogelijk moet zijn om de uitvoer van Nexpert te vertalen naar Visual Basic, kunnen we ons algoritme proberen te vertalen. Dit is meer bedoeld om aan te tonen dat de objectgeörienteerdheid en het redeneermechanisme te implementeren zijn.

De functies en procedures

'Bepaal_ploegen' in Nexpert notatie

Onze kennisregels zijn nog informeel en dat is niet onder te brengen in Nexpert. Dit zal dus eerst moeten veranderen, voordat we de vertaling naar Visual basic kunnen realiseren. Van een aantal commando's die gebruikt zijn, komt de syntax misschien niet helemaal overeen met die van Nexpert, maar de betekenis blijkt wel duidelijk.
We werken met verzamelingen objecten (werknemerspool en concentratiepool) en een verzameling afstanden (postcodepool). Werknemerspool bevat alle geselecteerde werknemers en hun postcodes. Wanneer we een concentratie vinden, creëeren we een concentratiepool. Aan het einde hebben we dus een aantal concentraties, in verschillende verzamelingen.

```
SUGGEST init

IF      =       werknemerspool.discipline"timmerman"
THEN            init
        =       afstand                 5
        =       postcodeafstand         0
DO              bepaal_ploegen
```

De rest van de regels van het algoritme zijn opgenomen in bijlage D. Aan de hand van deze regels moeten we een vertaling kunnen maken naar Visual Basic.

De vertaling

In eerste instantie zal de vertaling niet direct in Visual Basic kunnen draaien, omdat bijvoorbeeld voor het samenstellen van de verzamelingen een procedure geschreven moet worden. Verder valt op dat de code nogal veel rommel bevat in de vorm van veel if...then statements. Het is nu echter nog niet van belang om direct optimale code te genereren. De onderstaande functie is een vertaling van de regel uit de vorige paragraaf. De rest van de regels zijn opgenomen in bijlage D.

```
Function init () As Boolean
        If werknemers.discipline = "timmerman" Then
                Call MaakVerzameling(werknemers, werknemerspool)
        Else
                init = False
                Exit Function
        End if
        init = True
        afstand = 5
        postcodeafstand = 0
        If bepaal_ploegen Then
        End if
End Function
```

Zo zal de implementatie er grofweg uitzien in Visual Basic. Er zijn een aantal procedures geïntroduceerd die ongeveer doen wat de naam impliceert. Deze moeten echter nog wel geschreven worden.
De bovenstaande functies roepen elkaar aan, waardoor het principe van backward chaining in werking is. Forward chaining is er echter nog niet.

Forward chaining

Zoals we al zagen in hoofdstuk vijftien, moeten we van elke functie in een lijst gaan bijhouden in welke andere functies hij voorkomt. Nadat een functie nu True blijkt, roepen we aan de hand van die lijst de procedure **call_function aan** die aangeeft welke andere functies moeten volgen. Deze procedure-aanroep komt op het moment dat een functie binnen een andere functie gevalideerd is, maar de huidige functie doodloopt. Kortgezegd wordt de lege plaats in de if..then statements opgevuld met een variabele die aangeeft dat op het moment dat een volgende If regel faalt, de procedure call_function aangeroepen moet worden. Bijvoorbeeld:

```
Function postcode_geschikt () As Boolean
        Dim bepaal_afstand_bool As Boolean

        If bepaal_afstand Then
                bepaal_afstand_bool = True
        Else
                postcode_geschikt = False
                Exit Function
        End If
        If postcodeafstand < afstand Then
        Else
                postcode_geschikt = False
                Call call_function(FunctieLijstBepaalAfstand)
                Exit Function
        End If
        postcode_geschikt = True
        Call MaakVerzameling(werknemerspool.postcode, concentratiepool)
        Call Next (werknemerspool.postcode)
Exit Function
```

Conclusie

Wanneer we de vertaling nu goed bekijken en optimaliseren, kunnen we deze gegevens meenemen in het ontwikkelen van de parser. Er blijkt nu nog een hoop te optimaliseren in de programmacode, maar dat hoeft op zich geen belemmering te zijn voor de snelheid van de uitvoering. Duidelijk is wel dat er een aantal functionaliteiten binnen Nexpert zijn, die bepaalde standaardprocedures vereisen. Deze zullen dus eenmalig ontwikkeld moeten worden.

Hoofdstuk zeventien

Conclusies

De ontwikkeling van PlanMate werd gekenmerkt door het maken keuzes waarvan we niet wisten wat het effect van de mogelijke oplossingen zou zijn. Hierdoor werd bij het aandragen van oplossingen vaak gekeken naar de oplossing van gelijksoortige problemen. Dit was niet altijd even makkelijk, omdat er weinig moderne planningsapplicaties bestaan die de specifieke personeelsplanning in bouw kunnen ondersteunen. Mede daarom is het moeilijk te zeggen in hoeverre bepaalde keuzes goed of fout geweest zijn. Dit zal moeten blijken op het moment dat de gebruikers het pakket gaan gebruiken.

Oplossingen

Was er vraag naar planningsondersteuning?

Tijdens het gehele traject dat we met de ontwikkeling van PlanMate afgelegd hebben, werd steeds duidelijker dat er een grote vraag naar planningsondersteuning voor de personeelroostering bestaat. Dit komt deels doordat er weinig software beschikbaar is een deels omdat er steeds efficienter gewerkt moet gaan worden. Met deze informatie was het mogelijk om de ontwikkeling van PlanMate voort te zetten.

Welke rol speelde PlanMate I?

PlanMate I bleek een goede aanzet voor de ontwikkeling van Planmate II, maar kon niet als basis dienen voor het te ontwikkelen gebruikersinterface. Dit kwam voornamelijk doordat KPD de ontwikkelomgeving Visual Basic gebruikt, en PlanMate I teveel gericht was op de planningssituatie van MOES.
Wel diende het pakket als inspiratiebron. Het is alleen jammer dat er nooit feedback van de gebruiker (MOES) is gekomen door de problemen waarmee de installatie van Planmate I gepaard ging. Deze feedback is later wel verkregen door een ronde van prototyping van Planmate II. De informatie die hieruit verkregen werd, bleek vooral erg nuttig bij de ontwikkeling van het gebruikersinterface.

Hoe pasten we planMate II in de bestaande planningssituaties?

Zoals net al genoemd was de opzet één van de oorzaken waarom PlanMate I slechts als inspiratiebron werd gebruikt. Het gebruikersinterface was teveel gericht op de planningssituatie bij bouwbedrijf MOES.
Na een onderzoek naar de bestaande planningsstructuren bij de vier bouwbedrijven bleek dat de positie van PlanMate tussen de werkvloer en het planningsoverleg zou liggen. Dit was namelijk de plaats die in alle participerende bouwbedrijven naar voren kwam als de plek waar ondersteuning nodig was.

Hoe wilden de roostermakers ondersteund worden?

Bij de behoeftenanalyse toonden de behoeften van de roostermakers een zodanig verschil, dat het mogelijk moest zijn de administratieve mogelijkheden naar eigen wens uit te breiden. Daarom is een relationele database ontwikkeld waarin de roostermaker zijn eigen kwalificaties aan werknemers kan toevoegen.
Uit de behoeftenanalyse bleek verder de vraag naar rapportagemogelijkheden voor de roosterexpert zelf, maar ook voor de bedrijfstop. Deze waren vooral gericht op de mogelijkheid te controleren of de continuiteit en personeelsbezetting wel optimaal waren. Door deze rapportages is het mogelijk tijdig stappen te ondernemen wanneer er in de toekomst pieken of dalen dreigen op te treden in de vraag naar arbeid.
Daarnaast was een makkelijk en intuitief gebruikersinterface natuurlijk ook een vereiste, zeker omdat de roosterexperts die meewerkten, weinig tot geen ervaring hadden met de grafische werkomgeving die windows biedt.

Was het mogelijk een kennissysteem te ontwikkelen?

De behoeftenanalyse was tegelijkertijd de kennisacquistie die nodig was voor de ontwikkeling van controleur, evaluator en generator. De eerste twee zijn intussen geimplementeerd en zorgen voor de controle en evaluatie van een aantal belangrijke randvoorwaarden en doelfuncties. We onderscheiden er een aantal:

- Controle op CAO-regelgeving
- Controle op onmogelijkheden, zoals één persoon tegelijkertijd op meerdere plekken.
- Controle op andere onregelmatigheden in het personeelsrooster.
- Evaluatie op continuiteit van de personeelsleden
- Evaluatie op een minimale reisafstand tussen de werknemers en het werk
- Evaluatie op de continuiteit van de totale personeelsvraag.

De generator is nog in ontwikkeling, maar zal een beperkte functionaliteit kennen. Daarbij zal gebruik gemaakt worden van de bovenstaande randvoorwaarden en doelfuncties.

Hoe ontwikkelden we een goed gebruikersinterface?

Het gebruikersinterface dat uiteindelijk ontwikkeld is, is na prototyping bij de gebruikers tot stand gekomen. Hiermee is nog niet gezegd dat het in praktijk goed zal functioneren, omdat de roostertaak er nog niet continu mee wordt uitgevoerd. Wel bleek de prototyping erg nuttig. Naderhand kan ik concluderen dat het in een vroegtijdig stadium ontwikkelen van een gebruikersinterface een vereiste is bij de ontwikkeling van een software-applicatie.
Daarnaast blijkt de modellering via de OOSE-methode van Jacobson een zodanige, dat vanuit de gebruiker gekeken word hoe de software bediend moet gaan worden. Dit sloot goed aan bij de gesprekken die gevoerd zijn om de functionaliteit van Planmate te achterhalen.
De roosterexperts waar Jan Glaudé en ik contact mee hadden bleken weinig ervaring met computers te hebben, waardoor er nog meer aandacht aan het gebruikersinterface geschonken werd, en het nodig was de handleiding uitgebreider te maken. Gevaar hierbij was dat de meer geoefende gebruiker geïrriteerd kan raken door de makkelijke, maar vaak niet snelle manier van bedienen. Daarnaast zal hij of zij de handleiding niet goed doorlezen omdat er teveel tekst nodig is om zijn of haar specifieke probleem uit te leggen.

Konden we een vertaler voor de kennisbanken uit Nexpert ontwikkelen?

Een vertaling van de Nexpert-uitvoer naar een programma in een hogere programmeertaal lijkt tot de mogelijkheden te behoren. De uitvoer van Nexpert is duidelijk, overzichtelijk en bevat genoeg informatie. De implementatie van de objectgeorienteerdheid en het mechanisme van redeneren waren zaken die nog wel aangebracht moesten worden in de code. Doordat de structuren van forward chaining en backward chaining te vertalen zijn naar structuren in Visual Basic, was de vertaling mogelijk. de objectgeorienteerdheid werd toegevoegd door de properties van de klassen onder te brengen onder de objecten.
Er is gekozen voor een tussenstap tussen de scanner en de parser, zodat er gebruik kan worden gemaakt van dezelfde scanner bij een vertaling naar verschillende andere programmeertalen.

Laatste opmerkingen

Na het hele traject (op de verkoop na) afgelegd te hebben, heb ik vooral geleerd welke zaken belangrijk zijn in het ontwikkelproces, en welke minder tijd nodig hebben. In het begin is het vooral zaak snel een beeld te krijgen hoe de planningssituatie eruit ziet. Door alleen een literatuurstudie of door andere planningspakketten te bestuderen, krijg je dit beeld niet goed. Daarom is het erg nuttig de gebruikers met een gebruikersinterface te laten spelen. Hierdoor wordt namelijk goed duidelijk welke handelingen belangrijk zijn, en komt er een nieuw stuk functionaliteit naar boven. De implementatie loopt dan meer parallel met de acquisitie, waardoor je als ontwikkelaar een beter beeld krijgt over de planningssituatie en routines die daar voor geschreven moeten worden beter kan laten aansluiten.
De overstap naar het bedrijfsleven is er één waarbij je goed moet weten wat er van je gevraagd wordt. Vooral bij KPD was dat een hele opgave en is dat nu nog niet geheel duidelijk. Daarnaast ben je geheel afhankelijk van de verkopers in hoeverre jouw pakket verkocht gaat worden en hangt jouw baan dus gedeeltelijk af van anderen.
Persoonlijk ben ik er van overtuigd dat een stage, zoals deze er één was, een zeer nuttig en noodzakelijk gedeelte van je studie is. Ik kan het dus alleen maar aanbevelen aan andere studenten.

BIJLAGE A

Belangrijkste tabellen in PlanMate II

Personen			
Mannr	Text	20	
Naam	Text	50	
Ploeg	Text	20	
Adres	Text	50	
Postcode	Text	7	
Woonplaats	Text	50	
Vaardigheid	Text	100	
Datum in dienst	Double		
GeboorteDatum	Double		
Uurloon	Valuta		
Prestatie	Valuta		
NotVisible1	Text	255	
Projecten			
Pcode	Text	2	
Naam	Text	50	
Adres	Text	50	
Postcode	Text	7	
Plaats	Text	50	
Bewerkingen			
Bcode	Text	12	(Pcode + Bcode)
Naam	Text	50	
Behoefte	Single		
Discipline	Text	50	
Rooster			
Bcode	Text	12	
Mannr	Text	20	
Afstand	Double		
Tijd	Double		
Curs			
Naam	Text	50	
Cnummer	Double		
Select	Boolean		
Pe_Cu			
Mannr	Text	20	
Nummer	Double		
Disc			
Naam	Text	50	
Nummer	Double		
Select	Boolean		
Color	Long		
Pe_Di			
Mannr	Text	20	
Nummer	Double		
Be_Di			
Bcode	Text	20	

Nummer	Double	
<u>Kwal</u>		
Naam	Text	50
Nummer	Double	
Select	Boolean	
<u>Pe_Kw</u>		
Mannr	Text	20
Nummer	Double	

BIJLAGE B

Informele regels uit de kennisacquisitie

Geproduceerd door Jan Glaudé
Versie: 0.2

Grootel

Jan Schriever

ALS
het bouwplaatspersoneel is beschikbaar (komt vrij van een ander projekt)
& het bouwplaatspersoneel beschikt gegeven de behoefte over de juiste vakbekwaamheid
& het bouwplaatspersoneel kan de gecompliceerdheid van de bewerking aan
DAN
het bouwplaatspersoneel kan gekozen worden voor inroostering op de bewerking

ALS
het bouwplaatspersoneel kan gekozen worden voor inroostering op de bewerking
DAN
de personeelsplanner probeert de reiskilometers/-uren te minimaliseren

ALS
er wordt bouwplaatspersoneel gezocht,
& de discipline van het bouwplaatspersoneel is niet metselaars
DAN
het bouwplaatspersoneel wordt gezocht binnen de eigen regio

ALS
er wordt bouwplaatspersoneel gezocht,
& de discipline van het bouwplaatspersoneel is niet metselaars
& het bouwplaatspersoneel wordt niet gevonden in de regio
DAN
het bouwplaatspersoneel wordt gezocht buiten de eigen regio

ALS
er wordt bouwplaatspersoneel gezocht,
& de discipline van het bouwplaatspersoneel is metselaars
DAN
het bouwplaatspersoneel wordt gezocht zowel binnen als buiten de eigen regio

ALS
er wordt bouwplaatspersoneel gezocht
& het bouwplaatspersoneel wordt zowel binnen als buiten de eigen regio niet gevonden
DAN
er wordt gekeken naar andere bouwbedrijven voor inleen

ALS

het bouwplaatspersoneel heeft de funktie van leerling
DAN
het bouwplaatspersoneel wordt extra, niet vervangend, gepland

ALS
er is binnen een ploeg bouwplaatspersoneel geen EHBO'er aanwezig
DAN
tenminste 1 persoon uit de ploeg bouwplaatspersoneel krijgt een EHBO cursus

ALS
de bewerking metselen loopt uit
& de bewerking metselen mag niet uitlopen
DAN
eigen metselaars bijzetten voor 2 à 3 weken
& zoeken naar eigen metselaars buiten de eigen regio is toegestaan

ALS
de bewerking metselen loopt uit
& de bewerking metselen mag niet uitlopen
& eigen metselaars bijzetten voor 2 à 3 weken is niet mogelijk
DAN
er wordt geprobeerd metselaars bij te huren

Moes

Renée Schettler

ALS
de reisafstand van het woonadres van bouwplaatspersoneel tot de bewerking is te groot
& de duur van de bewerking is relatief lang (langer dan ca. 3 weken)
DAN
het aannemen van bouwplaatspersoneel voor de duur van het project is het meest geschikt

ALS
de reisafstand van het woonadres van bouwplaatspersoneel tot de bewerking is te groot
& de duur van de bewerking is relatief kort (korter dan ca. 3 weken)
DAN
het tijdelijk inhuren van bouwplaatspersoneel voor de duur van het project is het meest geschikt

ALS
aannemen of tijdelijk inhuren van bouwplaatspersoneel voor de duur van het project is het meest geschikt
& tijdelijk aannemen of tijdelijk inhuren van bouwplaatspersoneel in de buurt van het project is niet haalbaar
DAN
binnen het eigen bouwbedrijf wordt gekeken of er toch bouwplaatspersoneel voor de bewerking te vinden is

ALS
aannemen of tijdelijk inhuren van bouwplaatspersoneel voor de duur van het project is het meest geschikt
& tijdelijk aannemen of tijdelijk inhuren van bouwplaatspersoneel in de buurt van het project is niet haalbaar
& binnen het eigen bouwbedrijf is er geen bouwplaatspersoneel voor de bewerking te vinden
DAN
de bewerking wordt uitbesteed

& de bewerking hoeft verder niet te worden gepland

ALS
er is vraag naar bouwplaatspersoneel, uit een bepaalde discipline, voor een aantal weken
DAN
er wordt bepaald welk bouwplaatspersoneel gezien de beschikbaarheid, de discipline en de regio's waarin het woont in aanmerking komt

ALS
het bouwplaatspersoneel rijdt mee naar de bewerking in auto of bus
DAN
het bouwplaatspersoneel krijgt maximaal 7,5 reisuren per week vergoed, volgens de omrekening die in de bouw CAO is geregeld

Moes

Henk Jurjens

ALS
er moet een selektie plaatsvinden van te ontslagen bouwplaatspersoneel
DAN
de selektie vindt plaats op grond van de duur van het dienstverband van het bouwplaatspersoneel

ALS
er is een tekort aan bouwplaatspersoneel voorzien
& de bewerking ligt relatief ver weg van het woonadres van het bouwplaatspersoneel
& de bewerking duurt relatief kort
DAN
de bewerking wordt uitbesteed in de regio van de bewerking

ALS
er is een tekort aan bouwplaatspersoneel voorzien
& de bewerking ligt relatief ver weg van het woonadres van het bouwplaatspersoneel
& de bewerking duurt relatief kort
& de bewerking kan niet worden uitbesteed in de regio van de bewerking
& na afloop van de bewerking is ook nog werk te doen (3 à 6 maanden)
DAN
bouwplaatspersoneel wordt in de regio van de bewerking aangenomen

ALS
er is een overschot aan bouwplaatspersoneel voorzien
DAN
er wordt geprobeerd extra bouwplaatspersoneel op bewerking(en) te plaatsen

ALS
er is een overschot aan bouwplaatspersoneel voorzien
& plaatsen van extra bouwplaatspersoneel op bewerking(en) is niet mogelijk of gewenst
DAN
er wordt geprobeerd het overschot aan bouwplaatspersoneel uit te lenen aan andere bouwbedrijven

ALS

er is een overschot aan bouwplaatspersoneel voorzien
& plaatsen van extra bouwplaatspersoneel op bewerking(en) is niet mogelijk of gewenst
& uitlenen van bouwplaatspersoneel aan andere bouwbedrijven is niet mogelijk of gewenst
DAN
er wordt geprobeerd het overschot aan bouwplaatspersoneel te ontslaan

ALS
er wordt geprobeerd het overschot aan bouwplaatspersoneel te ontslaan
(en er is een overzicht van bouwplaatspersoneel die nu op bewerkingen gepland zijn plus een overzicht van het overschot aan bouwplaatspersoneel)
& de ontslagtermijn van het bouwplaatspersoneel is relatief kort
& de discipline van het bouwplaatspersoneel behoort tot de discipline van het overschot
& het bouwplaatspersoneel hoeft op een werk niet te worden vervangen door ander bouwplaatspersoneel
DAN
het bouwplaatspersoneel komt voor ontslag in aanmerking

ALS
de reistijd van de woning van het bouwplaatspersoneel naar de bewerking is langer dan anderhalf uur enkele reis
DAN
er wordt geprobeerd het bouwplaatspersoneel niet op de bewerking te plannen

ALS
er is een overschot aan bouwplaatspersoneel uit de renovatie
DAN
het overschot aan bouwplaatspersoneel uit de renovatie kan in beginsel de bouw in

ALS
het projekt is renovatie bewoonde toestand
& het bouwplaatspersoneel beschikt niet over sociale vaardigheden
DAN
het bouwplaatspersoneel kan niet op het projekt renovatie bewoonde toestand worden ingezet

ALS
het gevraagde bouwplaatspersoneel is restauratie bouwplaatspersoneel
DAN
er wordt gezocht naar beschikbaar restauratie bouwplaatspersoneel

ALS
het bouwplaatspersoneel is restauratie bouwplaatspersoneel
DAN
het bouwplaatspersoneel kan niet worden vervangen door niet restauratie bouwplaats-personeel

ALS
het bouwplaatspersoneel is bouw- en onderhoudservice bouwplaatspersoneel
DAN
het bouwplaatspersoneel kan niet worden ingezet op niet bouw- en onderhoudservice werkzaamheden

ALS
het gevraagde bouwplaatspersoneel is restauratie bouwplaatspersoneel
& restauratie bouwplaatspersoneel is niet beschikbaar
DAN
er wordt gezocht naar beschikbaar renovatie bouwplaatspersoneel

ALS
het gevraagde bouwplaatspersoneel is restauratie bouwplaatspersoneel
& beschikbaar renovatie bouwplaatspersoneel is niet gevonden
& beschikbaar restauratie bouwplaatspersoneel is niet gevonden
DAN
er wordt geprobeerd de restauratie bewerking(en) uit te besteden

ALS
binnen een bewerking is er vraag naar bouwplaatspersoneel met een bepaalde kwalificatie
DAN
er wordt geprobeerd bouwplaatspersoneel op de bewerking te zetten dat niet (veel) meer kwalificaties heeft dan de gevraagde kwalificaties

ALS
een ploeg bouwplaatspersoneel heeft een vaste samenstelling
DAN
de ploeg bouwplaatspersoneel kan niet uit elkaar worden gehaald

BIJLAGE C

Programmatekst: lijst van vraag op bewerkingen

```
Sub VraagLijst (Na#, Voor#)
  Dim Bool%, dDummy%
  Dim SQLQ$, bewerkingCode$, Bewerking$, sDiscipline$
  Dim Vraag#, BewNa#, BewVoor#, Behoefte#
  Dim dsBewerkingen As Dynaset, dsStroken As Dynaset

  SQLQ = "SELECT DISTINCT Bewerkingen.* FROM Bewerkingen,BStroken WHERE "
  SQLQ = SQLQ & " NOT (BStroken.start > " & Str$(Voor)
  SQLQ = SQLQ & " AND BStroken.eind < " & Str$(Na)
  SQLQ = SQLQ & ") AND Bstroken.BCode = Bewerkingen.BCode "
  Set dsBewerkingen = PlanMate.CreateDynaset(SQLQ)

  If dsBewerkingen.RecordCount <> 0 Then
    dsBewerkingen.MoveFirst
    VraagTekst = "Vraag" & Chr$(13) + Chr$(10) & Chr$(13) + Chr$(10)
    While Not dsBewerkingen.EOF
      Call DynasetAccess1(OP_LEZEN, dsBewerkingen, "BCode", bewerkingCode)
      Call DynasetAccess1(OP_LEZEN, dsBewerkingen, "Discipline", sDiscipline)
      Call DynasetAccess1(OP_LEZEN, dsBewerkingen, "Naam", Bewerking)
      Call DynasetAccess3(OP_LEZEN, dsBewerkingen, "Behoefte", Vraag)
            Set dsStroken = PlanMate.CreateDynaset("SELECT * FROM BStroken WHERE BCode = '" &
bewerkingCode & "' ORDER BY Start")
      If dsStroken.RecordCount <> 0 Then dsStroken.MoveFirst
      Behoefte = 0
      While Not dsStroken.EOF
        Call DynasetAccess3(OP_LEZEN, dsStroken, "Start", BewNa)
        Call DynasetAccess3(OP_LEZEN, dsStroken, "Eind", BewVoor)
        Call VraagBepalen(BewNa, BewVoor, bewerkingCode, Vraag, Behoefte)
                If Behoefte > 0 Then Call VraagLijstTekst(BewNa, BewVoor, bewerkingCode, sDiscipline,
Bewerking, Behoefte / Vraag)
        dsStroken.MoveNext
      Wend
      dsStroken.Close
      dsBewerkingen.MoveNext
    Wend
  Else
    VraagTekst = "Er is geen vraag ! "
  End If
  dsBewerkingen.Close
End Sub


Sub VraagBepalen (Na#, Voor#, Bewerking$, Vraag#, Behoefte#)
  Dim BewNa#, BewVoor#, dAanbod#, Strookna#, StrookVoor#
  Dim SQLQ$, sManNummer$
  Dim dsProcenten As Dynaset, dsStroken As Dynaset

  Behoefte = Int(Voor - Na) * Vraag
  SQLQ = "SELECT DISTINCT * FROM RStroken WHERE BCode = '" & Bewerking & "' AND "
  SQLQ = SQLQ & " NOT (start > " & Str$(Int(Voor)) & " AND eind < " & Str$(Int(Na))
  SQLQ = SQLQ & ") ORDER BY Mannr,Start ASC"
  Set dsStroken = PlanMate.CreateDynaset(SQLQ)
```

```
  If dsStroken.RecordCount <> 0 Then
    dsStroken.MoveFirst
    While Not dsStroken.EOF
      Call DynasetAccess3(OP_LEZEN, dsStroken, "Start", Strookna)
      Call DynasetAccess3(OP_LEZEN, dsStroken, "Eind", StrookVoor)
      Call DynasetAccess1(OP_LEZEN, dsStroken, "Mannr", sManNummer)
      SQLQ = "SELECT DISTINCT * FROM Procenten WHERE Mannr = '" & sManNummer & "' "
        SQLQ = SQLQ & " AND BCode = '" & Bewerking & "' AND Start >= " & Str$(Int(Strookna)) & " AND
Start < " & Str$(Int(StrookVoor))
      SQLQ = SQLQ & " ORDER BY Start ASC"
      Set dsProcenten = PlanMate.CreateDynaset(SQLQ)
      If dsProcenten.RecordCount <> 0 Then dsProcenten.MoveFirst
      While Not dsProcenten.EOF
        Call DynasetAccess3(OP_LEZEN, dsProcenten, "Start", BewNa)
        Call DynasetAccess3(OP_LEZEN, dsProcenten, "Procent", dAanbod)
        dsProcenten.MoveNext
        If Not dsProcenten.EOF Then
          Call DynasetAccess3(OP_LEZEN, dsProcenten, "Start", BewVoor)
        Else
          BewVoor = StrookVoor
        End If
        Behoefte = Behoefte - ((BewVoor - BewNa) * (dAanbod / 100))
      Wend
      dsProcenten.Close
      dsStroken.MoveNext
    Wend
  End If
  Behoefte = Int((Behoefte / (Voor - Na)) * 100)
  dsStroken.Close
End Sub


Sub VraagLijstTekst (Na#, Voor#, ProjectCode$, Discipline$, Bewerking$, Behoefte#)
  Dim sProjectNaam$
  Static sOProjectNaam$
  Dim dsProjecten As Dynaset


  Set dsProjecten = PlanMate.CreateDynaset("projecten")
  dsProjecten.FindFirst " Pcode = '" & Left$(ProjectCode, 2) & "' "

  Call DynasetAccess1(OP_LEZEN, dsProjecten, "Naam", sProjectNaam)
  If sProjectNaam = sOProjectNaam Then
    sProjectNaam = ""
  Else
    sOProjectNaam = sProjectNaam
  End If
  Call VraagTekstMaken(Na, Voor, sProjectNaam, Discipline, Bewerking, Behoefte)
  dsProjecten.Close
End Sub
```

BIJLAGE D

Vertaling algoritme naar Visual Basic

De Nexpert notatie

```
IF      <       afstand                         20
THEN            bepaal_ploegen
DO      =       beginpostcode                   werknemerspool.postcode
DO              vind_concentratie

IF      Yes     genoeg_postcodes_in_de_buurt
THEN            vind_concentratie
CREATE          concentratie                    concentratiepool
REMOVE          concentratie                    werknemerspool
DO              bepaal_ploegen

IF      Yes     bepaal_afstand
        <       postcodeafstand                 afstand
THEN            postcode_geschikt
PUT             werknemerspool.postcode         concentratiepool
NEXT            werknemerspool

IF      =       postcodepool.postcode1          beginpostcode
        =       postcodepool.postcode2          werknemerspool.postcode
THEN            bepaal_afstand
        =       postcodeafstand                 postcodepool.afstand

IF      Yes     postcode_geschikt
        >       #objecten(concentratiepool)     2
THEN            genoeg_postcodes_in_de_buurt

IF      NOT     vind_concentratie
        >       #objecten(werknemerspool)       2
THEN            andere_concentratie
FIRST           werknemerspool
        =       beginpostcode                   werknemerspool.postcode
DO              bepaal_ploegen

IF      NOT     vind_concentratie
        <       #objecten(werknemerspool)       3
THEN            geen_concentraties
        +       afstand                         5
FIRST           werknemerspool
        =       beginpostcode                   werknemerspool.postcode
DO              bepaal_ploegen
```

De vertaling in Visual Basic

```
Function bepaal_ploegen () As Boolean
        If afstand < 20 Then
        Else
                bepaal_ploegen = False
                Exit Function
```

```
        End If
        bepaal_ploegen = True
        beginpostcode = werknemerspool.postcode
        If vind_concentratie Then
        End If
Exit Function

Function vind_concentratie () As Boolean
        If genoeg_postcodes_in_de_buurt Then
        Else
                vind_concentratie = False
                Exit Function
        End If
        vind_concentratie = True
        Call MaakVerzameling(concentratie, concentratiepool)
        Call VerwijderUitVerzameling(concentratie, werknemerspool)
        If bepaal_ploegen Then
        End If
Exit Function

Function postcode_geschikt () As Boolean
        If bepaal_afstand Then
        Else
                postcode_geschikt = False
                Exit Function
        End If
        If postcodeafstand < afstand Then
        Else
                postcode_geschikt = False
                Exit Function
        End If
        postcode_geschikt = True
        Call MaakVerzameling(werknemerspool.postcode, concentratiepool)
        Call Next (werknemerspool.postcode)
Exit Function

Function bepaal_afstand () As Boolean
        If postcodepool.postcode1 = beginpostcode Then
        Else
                bepaal_afstand = False
                Exit Function
        End If
        If postcodepool.postcode2 = werknemerspool.postcode Then
        Else
                bepaal_afstand = False
                Exit Function
        End If
        bepaal_afstand = True
        postcodeafstand = postcodepool.afstand
Exit Function

Function genoeg_postcodes_in_de_buurt () As Boolean
        If postcode_geschikt Then
        Else
                genoeg_postcodes_in_de_buurt = False
                Exit Function
        End If
        If concentratiepool > 2 Then
        Else
                genoeg_postcodes_in_de_buurt = False
```

```
                Exit Function
        End If
        genoeg_postcodes_in_de_buurt = True
Exit Function

Function andere_concentratie () As Boolean
        If Not vind_concentratie Then
        Else
                andere_concentratie = False
                Exit Function
        End If
        If werknemerspool > 2 Then
        Else
                andere_concentratie = False
                Exit Function
        End If
        andere_concentratie = True
        Call First(werknemerspool)
        beginpostcode = werknemerspool.postcode
        If bepaal_ploegen Then
        End If
Exit Function

Function geen_concentratie () As Boolean
        If Not vind_concentratie Then
        Else
                geen_concentratie = False
                Exit Function
        End If
        If werknemerspool < 3 Then
        Else
                geen_concentratie = False
                Exit Function
        End If
        geen_concentratie = True
        afstand = afstand + 5
        Call First(werknemerspool)
        beginpostcode = werknemerspool.postcode
        If bepaal_ploegen Then
        End If
Exit Function
```

Literatuur

AND Software — AND Time/Distance server
1994, AND Software

Blaauw, A. — Project PPB, modellen voor personeelsplanning in de bouw
(1993), intern rapport

Cornell, G. — The Visual Basic 3 for Windows Handbook
1993, McGraw-Hill

Dean, T. — Artificial Intelligence, Theory and Practice
1995, Benjamin/Cunnings

Draper, N. — User centered system design
1988

Elmasri / Navathe — Fundamentals of Database Systems
1989, Benjamin/Cunnings

Glaudé, J.H. / Meer, R.K. v/d — PlanMate II
1994, intern rapport

Glaudé, J.H. — CAO bouwbedrijf 1 januari 1993 t/m 31 december 1994.
Produktieregels voor de planning van bouwplaatspersoneel
1994, intern rapport

Glaudé, J.H. — Produktieregels voor de planning van bouwplaatspersoneel
1994, intern rapport

Huisman, Jorna, Weezel — Algehele structuur voor het SEC
1994, intern rapport

Huisman, E. — Functionele specificatie en ontwerp Controleur en Evaluator
1994, IKS Projecten

Huisman, E. — Using EVGUI.DLL
1994, IKS Projecten

Jacobson, Ivar — Object-Oriented Software Engineering, A Use Case Driven Approach
1993, Addison-Wesley

Jorna, R. / Simons, J — Kennis in organisaties, toepassingen en theorie van kennissystemen
1992, Coutinho

Kamin S. — Programming Languages, An Interpreter Based Approach
1990, Addison-Wesley

McConnel, Steve — Code Complete, A Practical Handbook of Software Construction
1993, Microsoft Press

McDermit, J.A. — Software Engineer's Reference Book
1991

Meer, R.K. van der — PlanMate II, het behoeftenmodel
PlanMate II, het analysemodel
Rapportage bij de participerende bouwbedrijven
1994, interne rapporten

Numan, J.H. — Functionele beschrijving user-interface PlanMate
1993, intern rapport DISKUS (BPR-BS-UI-01-1.1)

Oldenkamp, J.H. Functioneel ontwerp planMate, 1992, (BPR-FO-SA-01-1.0)
Personeelsplanning voor bouwprojecten, 1992, (BPR-FO-HS-01-1.0)
Taakanalyse inroostering bouwplaatspersoneel, 1992, (BPR-FO-TA-01-1.0)
Plan van aanpak BPR, 1992, (BPR-FO-PA-01-0.1)
Intern rapporten DISKUS

Oldenkamp, J.H. Definities van kennissystemen
1993, Research rapport

Potter Visual Basic Superbible
1993, Waite Group Press

Preece, J. Human-Computer Interaction
1994, Addison-Wesley

Rich, E / Knight, K Artificial Intelligence
1991, McGraw-Hill

Sommerville, Ian Leerboek Software Engineering
1992, Addison-Wesley