University of Groningen **Faculty of Mathematics and Physical Sciences**

Department of Computing Science

WORDT
NIET UITGELEEND

NIET
UITLEEN-
BAAR

SIX

Expertsysteem voor de kwalitatieve analyse van het papierfabricageproces

Rik A. Carton & Mike Pruis

begeleiders: Prof.dr.ir. L. Spaanenburg
Drs. J. Jongejan

augustus 1995

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

RuG

SIX

EXPERTSYSTEEM VOOR DE KWALITATIEVE ANALYSE VAN HET PAPIERFABRICAGEPROCES

Rik A. Carton[1] & Mike Pruis[2]

Augustus 1995

- Vertis bv, Veendam
- Rijks*universiteit* Groningen,
  [1]Vakgroep Informatica, Faculteit der Wiskunde en Natuurwetenschappen
  [2]Vakgroep Alfa-Informatica, Faculteit der Letteren

VOORWOORD

Dit verslag is het resultaat van ons afstudeerproject bij Vertis in Veendam. Tijdens de afstudeerperiode, die ("ruim") zes maanden duurde, hebben we met een groot aantal mensen te maken gehad. Zonder hen zou het project ongetwijfeld anders zijn verlopen. Wij willen daarom de volgende mensen bedanken: Jelmer Sjollema (de begeleider bij Vertis), Mark Geerts (het hoofd van de afdeling ProMod van Vertis), Anne Hulst, Jan Hendriks en Jacob Terpstra (van het Papierlab van AVEBE in Foxhol), Gosse Bouma (de begeleider van Alfa-Informatica), Ben Spaanenburg en Jan Jongejan (de begeleiders van Informatica) en alle medewerkers van de afdeling ProMod van Vertis, die we om diverse redenen (onder andere voor het drinken van koffie en het afdrukken van "de nieuwste versie" van dit verslag) geregeld hebben lastiggevallen. Bedankt!

Rik Carton en Mike Pruis

SAMENVATTING

Vertis, een zelfstandige automatiseringsdochter van AVEBE, voert opdrachten uit voor zowel het moederbedrijf als andere bedrijven. De afdeling ProMod richt zich vooral op modellering van processen binnen de waterzuiveringswereld en de papierindustrie. Binnen deze afdeling is een prototype van een expertsysteem voor het wet-end deel van het papierfabricageproces ontwikkeld. Dit prototype heet SIX en geeft aan de hand van specificaties (meetwaarden) van een papierproces suggesties (voornamelijk betreffende zetmeel en andere additieven) voor optimalisatie van dat proces. De doelstelling van dit project was als volgt:

- Het prototype moet voldoende functionaliteit en mogelijkheden bezitten om (in de toekomst) een adviserende rol te gaan spelen in het Papierlab van AVEBE.
- Het prototype moet een voorbeeld zijn voor AVEBE van de mogelijkheden van dergelijke systemen binnen het papierfabricageproces of in het algemeen bij processen waarbij aardappelzetmeel gebruikt wordt.
- Het project moet voor Vertis kennis opleveren over het ontwikkelen van expert/adviessystemen in het algemeen en over de ontwikkelomgeving G2 in het bijzonder.

Op basis van de doelstelling kan gesteld worden dat er twee belangrijke aandachtspunten voor het prototype zijn. Ten eerste moeten de kennis en de redeneermethode op een inzichtelijke manier in het systeem worden opgenomen. Het systeem moet voor Vertis namelijk kunnen dienen als voorbeeld van een expertsysteem in G2. Tevens moet het kunnen worden uitgebreid tot een in de praktijk bruikbare applicatie (eventueel door overzetting naar een ander platform). Ten tweede is het vanwege de voorbeeldfunctie voor AVEBE cruciaal dat de interactie met de gebruiker op een prettige, eenvoudige wijze verloopt. Bovendien moet voor de papierexperts de kracht van een dergelijk systeem duidelijk worden.

Voor SIX is een beproefde methode voor de realisatie van expertsystemen gekozen: een regelgebaseerd produktiesysteem. Een belangrijke reden was het reeds bij Vertis aanwezige G2. G2 is een ontwikkelomgeving voor real-time expertsystemen op basis van regels. Tevens bleek tijdens de interviews met de experts dat dit de meest inzichtelijke representatie was voor de kennis (die voornamelijk in de vorm van vuistregels gehanteerd werd). Verder bleek tijdens deze interviews dat de vuistregels voornamelijk op vrij concrete numerieke waarden gebaseerd zijn, waardoor het niet noodzakelijk was om onzekerheid in het redeneren te verwerken. Het expertdeel van SIX is te karakteriseren als een diagnoseprobleem. De diagnose bestaat uit één of meer aan de hand van proceseigenschappen geconstateerde problemen in het proces, die kunnen worden uitgedrukt in verkeerd ingestelde procesparameters. Aan de hand van de gestelde diagnose kan worden geadviseerd de procesparameters te wijzigen (door middel van additieven). De kennis in het systeem bestaat uit domeinkennis (een gegevens– of feitenverzameling) en inferentiekennis (de regelverzameling). De inferentiekennis is onderverdeeld in drie lagen: classificatie– en initialisatie-kennis, diagnose-kennis en advieskennis. Het systeem redeneert voornamelijk op basis van bottom-up (data-gestuurde) inferentie. Vanuit de ingevoerde meetwaarden worden zoveel mogelijk conclusies afgeleid en waar nodig wordt een advies aan de gebruiker gepresenteerd.

De interface van SIX moet toegankelijk zijn. Deze eis kan in twee voorwaarden worden gesplitst: het programma moet een aantrekkelijk uiterlijk hebben en eenvoudig te bedienen zijn. Een visueel aantrekkelijke interface wekt gemakkelijk de belangstelling van de toekomstige gebruikers en kan bovendien helpen het leerproces te verkorten. Een systeem is eenvoudig te bedienen als het de gebruiker leidt, als het duidelijk maakt wat er op ieder moment gebeurt en als het tolerant met fouten van de gebruiker omspringt. Het geven van opdrachten moet zo eenvoudig mogelijk zijn, hetgeen voor de beginner of de leek betekent dat hij weinig zaken uit het hoofd hoeft te leren en op ieder moment kan zien welke opdrachten er gegeven kunnen worden. Ondanks het feit dat SIX de gebruiker niet leidt, voldoet het systeem redelijk goed aan deze eisen. Het is

daardoor vrij makkelijk toegankelijk, zodat de drempel tot het evalueren (en het gebruiken) van het systeem door het Papierlab van AVEBE laag is. Het hoofdscherm bestaat uit een schematische weergave van een papiermachine. De gebruiker kan meetwaarden invoeren door met de muis te "klikken" op één of meer onderdelen van de machine. Met dialoogvensters kunnen vervolgens drie typen gegevens worden ingevoerd. Na het invoeren activeert de gebruiker met één knop het redeneerproces van SIX.

SIX heeft in de huidige vorm nog enkele nadelen. Ten eerste zijn de adviezen (in de ogen van de experts) nog vrij voor de hand liggend. Dit is echter meestal het geval in het beginstadium van expertsystemen. Ten tweede geeft de interface niet de beoogde feedback, omdat de grafische ontwikkelhulp GUIDE/UIL niet snel genoeg is met het opbouwen van dialoogvensters. Ten derde is het prototype specifiek toegespitst op het papierfabricageproces. En ten vierde kunnen de door SIX gegeven adviezen niet worden beargumenteerd in de vorm van een afleiding.

Het verdient aanbeveling om SIX te veralgemeniseren. Hierdoor wordt het systeem makkelijker uitbreidbaar en zou het bijvoorbeeld ook in de textiel- en de levensmiddelenindustrie kunnen worden ingezet. Het uitbreiden van de kennis kan zorgen voor betere adviezen van het systeem.Verder lijkt het voor Vertis interessant om t.z.t. over te stappen naar versie 4.0 van G2 en GUIDE/UIL (en de daarvoor benodigde zwaardere machine aan te schaffen). Dit zal de responsproblemen van de interface grotendeels oplossen. Het beargumenteren van adviezen door het geven van een afleiding zal echter pas mogelijk zijn als G2 kan laten zien welke regels er zijn gebruikt om tot een bepaalde conclusie te komen.

Een interessante optie voor SIX is de uitbreiding tot een real-time expertsysteem, waarbij het programma wordt gekoppeld aan het papierfabricageproces en dus zelf de gegevens kan verzamelen. Als tussenstap (of als ondersteunende functionaliteit) zou SIX kunnen worden gekoppeld aan een model van het proces, dat dus de gegevens van het echte proces benadert of simuleert. Een koppeling van het expertsysteem en het proces zelf is echter de komende jaren nog niet te verwachten. Toch zou dit een zeer interessante uitbreiding zijn voor SIX.

ABSTRACT

The AVEBE daughter-company Vertis, which is an independent information technology firm, carries out tasks for AVEBE as well as other companies. The ProMod department emphasizes on modelling of water purification and papermaking processes. At this department, SIX, an expert system prototype has been developed, aimed at the wet-end part of the paper manufacturing process. SIX offers suggestions for optimising this process (mainly concerning starches and other additives) based on sample values. The goals for this project were:

- The prototype must contain sufficient functionality to act as an advisor for AVEBE's Paperlab (in the future).
- To AVEBE the prototype must be an example of the possibilities of such systems aimed at papermaking or at processes involving potato starch in general.
- The project must supply Vertis with knowledge on developing expert or advisory systems in general and on how to do this using the development tool G2 specifically.

Based on these goals two important points for attention stand out. Firstly, the knowledge and the inference method must be brought into the system in a comprehensible way, because it must serve as an example G2 expert system for Vertis. Also it must be possible to extend the system to be of use in practical situations (possibly by porting it to another platform). Secondly, because SIX must be an example to AVEBE, it is most important that the interaction between the user and the program is simple and pleasant. Moreover, the strength of such a system must be clear to the papermaking experts.

For SIX a well-tried method for realising expert systems has been chosen: a rule-based production system. An important reason for this choice was the availability of G2 at Vertis. This is a real-time expert system development tool. During the interviews with the experts it also became apparent that this would be the most comprehensible way to represent the knowledge (which was available mainly in the form of rules of thumb). These rules of thumb turned out to be based on fairly tangible numerical values, which excluded the need for reasoning with uncertainty. The expert part of SIX can be characterised as a diagnostic problem. The diagnosis consists of one or more problems established from process properties, that can be expressed as wrongly set process parameters. From the diagnosis the system can advise to change the parameters (using additives). The knowledge in the system consists of domain knowledge (a set of facts) and inference knowledge (the set of rules). The inference knowledge can be divided into three layers: classification and initialisation knowledge, diagnostic knowledge and advisory knowledge. The reasoning is based mainly on bottom-up inference (data-driven). As many conclusions as possible are drawn from the supplied sample values and the user is given advice where appropriate.

SIX' user interface must be accessible. This demand can be divided into two conditions, namely: the program must look attractive and must be easy to operate. A visually pleasing interface easily attracts the future users' attention and can help shortening the learning process. A system is easy to operate if it leads the user, if it shows what is happening at any point in time and if it tolerates users' mistakes. Issuing commands must be as straightforward as possible. Providing an overview of all possible commands will minimise the need for the uninitiated to learn commands by heart. Despite the fact that SIX does not lead the user, it meets the requirements reasonably well. Therefore it is fairly accessible and thus provides the AVEBE Paperlab with a low threshold for evaluating the system. The main window consists of a schematic representation of a papermachine. The user can enter sample values by "clicking" on one or more parts of the machine. Through the use of dialog boxes, three types of data can then be entered. When all data has been supplied the user activates the inference process with one button.

In its current form SIX has a number of disadvantages. Firstly, the advice is rather obvious (from the experts' point of view). This is however usually the case at the initial stage of an expert system. Secondly, the interface does not provide the intended feedback, because it takes the graphical development tool GUIDE/UIL too long to build up dialog boxes. Thirdly, the prototype is specifically aimed at the papermanufacturing process. Fourthly, the advice that SIX gives cannot be backed up by a derivation.

It is advisable to generalise SIX. By doing so, the system will become easier to extend and could be adapted for the textile industry and the foods industry. Extending the knowledge can lead to better advice. Furthermore it seems interesting for Vertis to upgrade to version 4.0 of G2 and GUIDE/UIL (and to the more powerful machine that these programs require) in the near future. This will largely solve the interface feedback problems. Underpinning the advice that SIX provides with a derivation however, will not be possible until G2 offers the facility to show the rules that have led to a certain conclusion.

An interesting option for SIX is extending it to a real-time expert system by linking it to the papermaking process and thus allowing it to collect the data by itself. As an intermediate solution (or for supporting functionality) SIX might be linked to a model of the real process, which simulates or approximates the real process data. A link between the expert system and the papermanufacturing process itself however, is not to be expected for a number of years. Yet it would make a very interesting extension for SIX.

INHOUDSOPGAVE

VOORWOORD V
SAMENVATTING VII
ABSTRACT IX
INHOUDSOPGAVE XI
1. INLEIDING 1
2. PROBLEEMSTELLING 3
2.1. Opdrachtomschrijving 3
2.1.1. Oorspronkelijke opdracht 3
2.1.2. Aangepaste opdracht 4
2.2. Doelstelling 5
3. PLANNING 7
3.1. Mini KADS 7
3.1.1. Modellen 7
3.1.2. Projectplanning binnen Mini KADS 8
3.2. Mini KADS en SIX 8
3.3. Projectplanning 8
4. PAPIERFABRICAGE 11
4.1. Fabricageproces 11
4.2. Additieven 12
4.3. Zetmeel 13
4.3.1. Zetmeel en zetmeelderivaten 13
4.3.2. Zetmeel ter verhoging van de interne sterkte 15
4.3.3. Zetmeel als retentiemiddel 16
4.4. Proceseigenschappen in het wet-end 17
5. PROBLEEMCONTEXT 19
5.1. Abstractie van het probleem 19
5.1.1. Expertgedeelte 19
5.1.2. Interface-gedeelte 20
5.2. Vergelijkbare expertsystemen 20
5.2.1. MYCIN 20
5.2.2. SPHINX 21
5.2.3. EPAK 22
6. PROBLEEMANALYSE 25
6.1. Oplossingsmethoden 25
6.1.1. Database 25
6.1.2. Neuraal netwerk 26
6.1.3. Fuzzy logic 26
6.1.4. Expertsysteem 27
6.1.5. Hybride systeem 27
6.2. Gebruikte methode 27

7. KENNISREPRESENTATIE EN EXPERTSYSTEMEN 29
7.1. Logica 29
7.1.1. Propositie– en predikatenlogica 29
7.1.2. Resolutie 32
7.2. Produktieregels en inferentie 32
7.2.1. Globale gegevensverzameling 33
7.2.2. Knowledge base 33
7.2.3. Inferentie in een produktiesysteem 33
7.2.4. Matching 35
7.2.5. Conflictresolutie 37
7.4. Onzekerheid bij het redeneren 39
7.4.1. Zekerheidsfactoren 40
7.4.2. Bayesiaanse netwerken 42
7.4.3. Dempster-Shafer theorie 44
8. DE USER INTERFACE 45
8.1. Voorwaarden voor de interface van SIX 45
8.2. Communicatie tussen mens en computer 46
8.3. Soorten user interfaces 47
8.3.1. Commando-interface 49
8.3.2. Natuurlijke taal-interface 50
8.3.3. Menu-georiënteerde interface 51
8.3.4. Grafische user-interface 52
8.3.5. Combinaties van interfaces 53
8.4. Keuze van de interface voor SIX 53
8.5. Evaluatie 54
9. GENSYM G2 55
9.1. Algemeen 55
9.2. Object-oriëntatie 57
9.3. Expertsysteem-ontwikkelomgeving 57
9.3.1. Produktieregels 57
9.3.2. Het inferentie-mechanisme 58
10. ONTWERP SIX 61
10.1. Kennisacquisitie 61
10.2. Kennisstructuur (expertisemodel) 61
10.2.1. Entiteiten en relaties (domeinlaag of gegevensbeschrijving) 62
10.2.2. De bouwstenen van een redenering (inferentielaag) 64
10.3. User interface 67
11. IMPLEMENTATIE SIX 73
11.1. User interface 74
11.2. Implementatie van de kennis 75
11.2.1. Kennisdefinitie 75
11.2.2. Kennisverzameling 77
12. RESULTATEN EN AANBEVELINGEN 81
BIJLAGE A: WOORDENLIJST 83
BIJLAGE B: INTERVIEWS 87
BIJLAGE C: VOORBEELDRAPPORT PAPIERLAB 89
BIJLAGE D: G2 LANGUAGE REFERENCE 91
LITERATUURLIJST 95
INDEX 97

1. INLEIDING

Dit verslag beschrijft de opzet en de ontwikkeling van *SIX*. SIX is een prototype van een *expertsysteem*[1] dat is gebouwd in opdracht van Vertis B.V. te Veendam. Vertis richt zich hiermee op haar moederbedrijf AVEBE, de grootste aardappelmeelproducent ter wereld. Een korte beschrijving van de beide bedrijven is hier op zijn plaats.

AVEBE fabriceert jaarlijks zo'n 500.000 ton aardappelzetmeel in onder andere Nederland, Duitsland, Frankrijk en Zweden. Bovendien exploiteert de firma samen met andere ondernemingen fabrieken in Thailand en in de Verenigde Staten. De verkoopkantoren van AVEBE, die verspreid zijn over de hele wereld, zorgen voor de afzet van het geproduceerde zetmeel. Daarnaast adviseert het bedrijf zijn klanten omtrent de keuze en het gebruik van zetmeel voor specifieke toepassingen. Het zetmeel vindt toepassing in onder andere levensmiddelen, textiel, farmaceutische produkten en papierprodukten.

Vertis is een automatiseringsbedrijf met ongeveer honderd werknemers. Het is zelfstandig, maar onderhoudt nauwe betrekkingen met AVEBE. Deze relatie is historisch bepaald. Vertis is namelijk ontstaan uit een joint venture tussen AVEBE en AKZO. In 1990 is ze zelfstandig geworden. Vertis voert nog steeds opdrachten uit voor AVEBE, maar richt zich ook op andere bedrijven. De meeste werknemers van Vertis hebben een universitaire opleiding wiskunde, informatica, chemie, procestechnologie of milieutechnologie gehad. De activiteiten van Vertis vallen ook onder deze vakgebieden en daarnaast wordt er veel op logistiek en administratief gebied gedaan.

SIX is bedoeld als adviserend systeem voor het Papierlab van AVEBE. AVEBE adviseert papierfabrikanten uit de gehele wereld over het gebruik van zetmeel in hun papierprodukten. Het onderzoek dat daarvoor nodig is, wordt door het Papierlab gedaan. De experts krijgen gegevens van het produktieproces bij een klant en doen bij AVEBE proeven waarbij ze bepaalde delen van het proces bij de klant simuleren. Door te redeneren over de resultaten en door bepaalde parameters in de proefopstelling te wijzigen, probeert men mogelijkheden te vinden tot verbetering van het proces bij de klant. In het ideale geval zou SIX een aantal van deze proeven overbodig maken. Het expertsysteem zou dan een efficiënt middel zijn om uit te vinden op welke manieren het proces van de klant kan worden verbeterd. Maar zover is het nog niet. SIX is in zijn huidige vorm een expertsysteem dat advies geeft over de verbetering het *wet-end*; dàt deel van de papierfabricage waar *pulp* wordt omgezet in papier.

SIX is het afstudeerproject van Rik Carton en Mike Pruis van de Rijksuniversiteit Groningen (RUG). We sluiten hiermee respectievelijk onze studies Informatica en Alfa-Informatica af. Het project is uitgevoerd bij de afdeling Procesmodellering (ProMod) van Vertis. De totstandkoming van het programma staat in dit verslag beschreven. Omdat het hier een gezamenlijk werk betreft, is aan het begin van een aantal hoofdstukken vermeld door wie de verschillende onderdelen van die hoofdstukken zijn geschreven.

De opbouw van het verslag is als volgt. In hoofdstuk 2 wordt de probleemstelling gegeven. Het volgende hoofdstuk bevat de planning voor het project. Dan volgt in hoofdstuk 4 een beknopte beschrijving van het papierfabricageproces, waarbij een aantal belangrijke zaken nader wordt toegelicht. Daarna behandelt hoofdstuk 5 expertsystemen die min of meer vergelijkbaar zijn met SIX. Het daaropvolgende hoofdstuk zet enkele oplossingsmethoden op een rij, die voor de *implementatie* van het expertsysteem zouden kunnen worden gebruikt. Dan volgt hoofdstuk 7, waarin een uiteenzetting wordt gegeven van *kennisrepresentatie* en expertsystemen. Hoofdstuk 8 geeft informatie over het ontwerpen van *user interfaces*, waarna in hoofdstuk 9 de *expertsysteem-shell* G2 aan bod komt. G2 is het systeem waarin SIX is geprogrammeerd. Na al deze zaken

[1] In dit verslag zullen de benamingen "expertsysteem", "kennissysteem" en "adviessysteem" doorelkaar worden gebruikt.

behandelen de hoofdstukken 10 en 11 respectievelijk het ontwerp en de implementatie van SIX. Tot slot worden in hoofdstuk 12 de resultaten en de aanbevelingen voor Vertis gegeven. Veel van de in dit verslag gebruikte termen worden verklaard in bijlage A. Deze termen worden de eerste keer dat ze in de tekst voorkomen *cursief* weergegeven. Bijlage B bevat de verslagen van de interviews die zijn gehouden met twee medewerkers van het Papierlab van AVEBE, bijlage C bevat een voorbeeldrapport met gegevens over het onderzoek naar de verbetering van een papierfabricageproces en bijlage D geeft een beknopt overzicht van de mogelijkheden van de taal G2.

2. PROBLEEMSTELLING

De probleemstelling die centraal staat in dit verslag wordt hier beschreven. In de eerste paragraaf staat de oorspronkelijke opdrachtomschrijving, zoals deze bij de aanvang van de afstudeerperiode is bepaald. Later is de opdracht aangepast en de nieuwe omschrijving wordt uiteraard ook gegeven. De tweede paragraaf behandelt vervolgens de doelstelling die uit de opdracht is voortgekomen.

2.1. Opdrachtomschrijving

De oorspronkelijke opdracht was gericht op het geven van advies over het gebruik van zetmeel bij de papierfabricage. De uiteindelijke opdracht richt zich niet alleen op zetmeel, maar op alle toevoegingen in het zogenaamde wet-end van het proces.

2.1.1. Oorspronkelijke opdracht

Bij de fabricage van papier wordt om verschillende redenen zetmeel als bestanddeel gebruikt. Het kan het papier bijvoorbeeld sterker maken en tijdens het fabricageproces zorgen voor een betere opname van stoffen in het papier. Papierfabrikanten betrekken (aardappel)zetmeel van AVEBE. Er zijn veel verschillende soorten zetmeel, voor verschillende doeleinden. Deskundigen van AVEBE bepalen met behulp van een aantal tests welk zetmeelprodukt het meest geschikt is voor de door de fabrikant beoogde toepassing. Omdat van te voren niet exact bepaald kan worden op welke manier het zetmeel gaat reageren met de andere papierbestanddelen, is de ervaring en know-how van deskundigen een belangrijke factor bij het kiezen van het produkt.

Het is voor AVEBE wenselijk deze ervaring en know-how vast te leggen, zodat de kennis binnen het bedrijf ook voor anderen dan alleen de deskundigen toegankelijk is. Verder zouden verkoopkantoren en vertegenwoordigers van AVEBE de klanten met behulp van deze kennis op efficiëntere wijze van advies kunnen voorzien. Vertis wil daarom de ervaring/know-how van de deskundigen vastleggen in een *kennissysteem*. Het systeem zou aan de hand van de gegevens van de klant (o.a. wensen en processpecificaties) en de in het systeem opgeslagen kennis een gefundeerde keuze moeten maken uit de zetmeelprodukten van AVEBE en eventueel alternatieven moeten geven.

Een dergelijk systeem zou uiteindelijk voor het gehele assortiment van AVEBE gebruikt kunnen worden. Het gaat dan om bijvoorbeeld zetmeel voor gebruik in textiel en levensmiddelen. AVEBE is momenteel bezig zich specialistischer en slagvaardiger op de markt te presenteren. Hierbij hoort ook het snel en flexibel kunnen inspelen op specifieke klantwensen. Een expertsysteem voor zetmeel zou hierbij een belangrijk hulpmiddel zijn.

Om het probleem te beperken wordt voor deze opdracht alleen gekeken naar de *zetmeelderivaten* die gebruikt worden in het wet-end van het papierproduktieproces. Er is voor dit deelgebied gekozen omdat de kennis erover bij het Papierlab van AVEBE het eenvoudigst expliciet te maken is. Tevens is er binnen de afdeling ProMod van Vertis een redelijke hoeveelheid kennis aanwezig op dit gebied. Door het systeem speciaal voor dit toepassingsgebied te maken, ligt een aantal keuzes in het systeem bij voorbaat vast, maar dit verandert niets aan het eigenlijke probleem. Het systeem kan later worden uitgebreid voor de andere gebieden. De tijdrovende *kennisacquisitie* voor deze gebieden kan echter binnen dit project achterwege worden gelaten.

Het eindprodukt van het project moet een prototype van een expertsysteem zijn, gerealiseerd in de expert-systeem-shell G2. Vertis wil dit prototype gebruiken om AVEBE de mogelijkheden van een dergelijk expertsysteem te demonstreren. Verder is het voor Vertis van belang dat er binnen het bedrijf (c.q. binnen de afdeling) ervaring wordt opgedaan met methoden om een expertsysteem te bouwen, zodat men in de toekomst beter op soortgelijke problemen is voorbereid.

De opdracht bestond uit het vastleggen en structureren van kennis en het bouwen van een prototype voor een expertsysteem. Het systeem moest als invoer een aantal gewenste *papiereigenschappen* ("papierwensen") krijgen en informatie over het huidige proces en de huidige papiereigenschappen . De uitvoer zou dan moeten bestaan uit de naam van een zetmeelderivaat, een plaats aanduiding in het proces en een bepaalde hoeveelheid. Het toevoegen van het bewuste zetmeelderivaat op de gegeven plaats in de gegeven hoeveelheid zou de gewenste papiereigenschappen tot gevolg moeten hebben. Verder moest het systeem de gemaakte keuzes kunnen verklaren. De activiteiten voor de opdracht zouden bestaan uit: het inwerken in G2 en het produceren van een overzicht van het kennis/expertdeel van G2, het vastleggen van de te gebruiken kennisstruc tuur en de structuur van de processpecificaties van de klant en het koppelen van deze structuren, het verzamelen van data en het inpassen ervan in de genoemde structuren en vervolgens het (gedeel telijk) implementeren van het systeem.

2.1.2. Aangepaste opdracht

Uit interviews met AVEBE bleek dat de papierwensen van de klant in de meeste gevallen nogal beperkt zijn. De papierproducent vraagt AVEBE vrijwel altijd naar een manier om de *retentie* van het papier te verhogen. SIX zou dus de mogelijkheid moeten bieden om een diversiteit aan gewenste wijzigingen in de papiereigenschappen te verwerken, terwijl slechts een klein deel daarvan echt gebruikt zou worden. Daarom is de opdracht aangepast.

De huidige doelstelling van SIX is het assisteren bij de optimalisatie van het wet-end proces in de papier-produktie. Hiervoor is gekozen naar aanleiding van de vragen van de klanten van AVEBE, die in de meeste gevallen een analyse van hun wet-end-proces willen. Zo'n analyse ondersteunt het zoeken naar een beter proces. Het woord "beter" kan daarbij bijvoorbeeld de volgende dingen impliceren:

- hogere zetmeelretentie
- snellere zetmeelretentie
- kleinere hoeveelheid milieu-onvriendelijke *additieven*
- kleinere hoeveelheid dure additieven

De invoer voor SIX moet bestaan uit de huidige eigenschappen van het wet-end proces. Deze eigenschappen bestaan uit een lijst van gegevens die op een aantal plaatsen in het wet-end worden gemeten. De metingen vinden meestal plaats na de toevoeging van een additief. In de praktijk krijgt AVEBE van de klant een aantal *pulp samples*, voorzien van enkele gemeten waarden en gegevens over het proces. AVEBE onderzoekt deze samples dan door allerlei metingen te doen, om zo alle voor hen relevante gegevens te achterhalen. De klant geeft ook aan AVEBE door wat hij[2] in zijn proces anders zou willen hebben. Vaak vraagt de klant alleen om "optimalisatie van het proces", maar in een aantal gevallen wordt er een specifiekere wens opgegeven. Voorbeelden hiervan zijn: verbetering van de zetmeelretentie en verhoging van de toegepaste hoeveelheid zetmeel. SIX richt zich in eerste instantie op de optimalisatie van het proces volgens de ideeën van AVEBE.

AVEBE heeft kennis over het optimaliseren van onder andere het wet-end-proces. Het grootste deel van deze kennis berust op ervaring. Een gedeelte van die kennis kan in standaardregels worden uitgedrukt die bijvoorbeeld aangeven onder welke omstandigheden zetmeel het best in het papier-in-wording wordt opgenomen. Door de kennis (standaardregels en ervaringskennis) in SIX op te nemen kan het systeem AVEBE assisteren in het optimaliseren van het wet-end bij de klant. De eigenschappen van de pulp die niet door de klant worden aangeleverd moeten nog wel door AVEBE worden bepaald, maar het aantal proeven dat nodig is om te bepalen welke wijzigingen er in het proces moeten worden aangebracht zal kleiner worden. SIX moet AVEBE in de toekomst in staat stellen na het verzamelen van gegevens over het bewuste wet-end-proces bij de klant direct advies aan deze klant uit te brengen. Hierdoor zal dus snelheidswinst behaald worden.

---

[2] Overal waar in de tekst "hij" staat, moet "hij of zij" worden gelezen.

De nieuwe opdracht is als volgt: SIX moet bepalen welke parameters van een bepaald wet-end-proces niet optimaal zijn en vervolgens naar een oplossing gaan zoeken. De oplossing zal bestaan uit een advies voor het vergroten of verkleinen van de dosering van een bepaald additief, of mogelijk het toepassen van een nieuw of vervangend additief. Ook moet worden vermeld op welke plaats in het wet-end de wijziging moet worden uitgevoerd. Het advies kan uit één verandering in de additieven bestaan, maar het is ook mogelijk dat er meerdere wijzigingen in het proces moeten plaatsvinden (op één of meer plaatsen) om het te verbeteren. Bovendien zal het advies van een beargumentering vergezeld moeten gaan, die de beweegreden van het systeem voor de betreffende keuze weergeeft.

2.2. Doelstelling

Naar aanleiding van de vorige paragraaf kan de doelstelling van SIX worden geformuleerd. In overleg met Vertis is uiteindelijk het onderstaande vastgelegd als doelstelling van het project:

Het doel van project SIX is een prototype (gerealiseerd in de ontwikkelomgeving G2) van een expert/adviessysteem voor het Papierlab van AVEBE. Het moet aan de volgende eisen voldoen:

- *Het prototype moet voldoende functionaliteit bezitten om een adviserende rol te spelen in het Papierlab. Aan de hand van specificaties van een wet-end papierproces geeft het systeem een suggestie om het proces te optimaliseren door middel van het doseren van additieven op een bepaalde plaats.*
- *Het prototype moet een voorbeeld zijn voor AVEBE van de mogelijkheden van dergelijke systemen binnen het papierfabricageproces of in het algemeen bij processen waarbij aardappelzetmeel wordt gebruikt.*
- *Het project moet voor Vertis ervaring/kennis opleveren over het ontwikkelen van expert/adviessystemen met behulp van de ontwikkelomgeving G2.*

---

3. PLANNING

Naar aanleiding van de in het vorige hoofdstuk opgestelde doelstelling is er bekeken welke onderdelen/taken er moesten plaatsvinden om het gestelde doel te halen. Aan de hand van deze lijst is er voor week 1 (januari) tot en met 26 (eind juli) een planning opgesteld. Om structuur in de planning aan te brengen en tegelijkertijd een ontwikkelmethode voor het project te hebben, is de planning opgesteld op basis van de Mini KADS methodiek. In dit hoofdstuk wordt eerst de Mini KADS methodiek besproken en wordt vervolgens de planning voor SIX toegelicht.

3.1. Mini KADS

Eén van de meest bekende methodes voor het ontwikkelen van kennissystemen heet KADS[3]. Een afgeleide daarvan is Mini KADS. Dit is een configuratie van de KADS -methode en is speciaal ontwikkeld voor de ondersteuning van kleinschalige projecten [Verschoor, 1992]. Kleinschalige projecten worden in Mini KADS als volgt gedefinieerd:

- de doorlooptijd is niet langer dan een jaar
- het aantal geïnvesteerde manjaren bedraagt niet meer dan een jaar
- het projectteam bestaat uit maximaal drie personen

Mini KADS is net als KADS een produktgedreven methode waarbij aan het eind van een project modellen en programmatuur opgeleverd worden. Binnen Mini KADS wordt onderscheid gemaakt tussen vier (groepen) modellen: het organisatiemodel, het taak-agent-model, het expertisemodel en het communicatie-ontwerp-model [Stemerdink].

3.1.1. Modellen

Het organisatiemodel geeft inzicht in de bedrijfskundige aspecten van het systeem. Het geeft weer wat de doelstellingen en randvoorwaarden zijn van alle betrokkenen bij het project. In het taak-agent-model worden taken en agenten gemodelleerd. Agenten worden gedefinieerd als *entiteiten* met bepaalde taken. De twee voor de hand liggende (en belangrijkste) agenten zijn het te bouwen expertsysteem en de gebruiker. Verder worden alle andere entiteiten die met het systeem zullen samenwerken gemodelleerd. Het expertisemodel is een beschrijving van alle kennisintensieve taken. De kennis wordt beschreven volgens het vierlagenmodel van KADS, waarvan er binnen Mini KADS drie worden uitgewerkt: de domeinlaag, de inferentielaag en de taaklaag (boven de taaklaag is in KADS nog de strategische laag gedefinieerd). Communicatietaken worden gemodelleerd in het communicatie-ontwerp-model. De communicatie tussen de gebruiker en het systeem kan bijvoorbeeld gegeven worden door een beschrijving van de gebruikersinterface en de lay-outs van rapporten, formulieren en dergelijke. De in– en output van het systeem vormen de schakel tussen het communicatie-ontwerp-model en het expertisemodel [Stemerdink].

In de domeinlaag uit de lagenstructuur van het expertisemodel wordt de kennis gerepresenteerd met behulp van hiërarchische gegevensmodellen (*E/R diagrammen*), beslisregels en *beslisbomen*. De inferentielaag geeft

[3] KADS staat voor Knowledge Acquisition, Documentation and Structuring system. Deze betekenis wordt echter tegenwoordig niet meer gebruikt.

inzicht in de structuur van de kennis. Met behulp van *DFD* (Data Flow Diagram)-achtige diagrammen worden de achtereenvolgende stappen in het probleemoplosproces gerepresenteerd. De derde laag (de taaklaag) brengt sturing aan in de inferentielaag.

3.1.2. Projectplanning binnen Mini KADS

De modellen zoals genoemd in de vorige paragraaf worden ten dele sequentieel ontwikkeld. Ze worden grotendeels parallel uitgewerkt en regelmatig herzien. Binnen Mini KADS wordt daarom niet gesproken over ontwikkelfasen maar over cycli. Het Mini KADS ontwikkeltraject bestaat uit vijf cycli, namelijk: projectidentificatie (cyclus 0), projectspecificatie (cyclus 1), systeemspecificatie (cyclus 2), systeembouw (cyclus 3) en gebruikerstest (cyclus 4) [Stemerdink].

Tijdens cyclus 0 wordt een mogelijk kansrijke expertsysteemtoepassing gesignaleerd. Tijdens cyclus 1 worden de projectdoelen onderzocht en gespecificeerd. Alle belanghebbenden worden geraadpleegd, zodat een goed beeld van de randvoorwaarden van het systeem gevormd wordt. Na afloop van deze cyclus wordt een projectvoorstel beschreven waarin een aantal vastgestelde onderwerpen aan bod komen. Het expertise- en communicatie-ontwerp-model worden in cyclus 2 tot in detail uitgewerkt. Tijdens deze cyclus vindt intensieve communicatie plaats tussen de ontwikkelaars en de "experts". De implementatie van het systeem vindt plaats in de derde cyclus. Deze cyclus levert een nulversie van het programma op die getest wordt bij de gebruikers en experts aan de hand van zoveel mogelijk praktijkgevallen. Tijdens de laatste cyclus vindt een uitgebreide test plaats met behulp van gebruikers die nog niet eerder bij het project betrokken waren. Deze tests leveren de informatie op die noodzakelijk is voor de afstemming van het communicatie-ontwerp-model en de programmatuur.

3.2. Mini KADS en SIX

Voor het SIX project gelden de volgende gegevens: de doorlooptijd is een half jaar, het aantal manjaren is één en de projectgroep bestaat uit 2 personen. Volgens de Mini KADS specificaties zou Mini KADS dus een geschikte methodiek zijn om dit project mee aan te pakken. Omdat SIX fungeert als een soort *pilot project* was voor aanvang van het project al duidelijk dat tijdens de ontwikkeling op bepaalde modellen en cycli de nadruk zou komen te liggen terwijl andere modellen en cycli bijna verwaarloosbaar waren.

Binnen Mini KADS wordt na cyclus 1 een projectvoorstel opgeleverd waaruit op verantwoorde wijze een voortgangsbeslissing kan worden genomen. Het voorstel bevat o.a. de volgende onderwerpen: projectdoelen, taak-agent-model (voorlopige versie), communicatiemodel (voorlopige schets), kennisbronnen, projectplanning en de specificatie van eind- en tussenresultaten. De punten die hier genoemd zijn zijn uit de complete lijst gekozen omdat ze ook binnen dit project van toepassing zijn. Aan het begin van het project is een projectvoorstel opgesteld waarin deze punten aanwezig waren. Het gebruik van Mini KADS voor SIX is vooral naar voren gekomen tijdens cyclus 1. Voor de rest van het project is wel de planning aangehouden maar is geen aandacht besteed aan het werken met behulp van de specifieke (Mini) KADS elementen. Binnen SIX heeft de Mini KADS methodiek vooral gefungeerd als plannings-leidraad. De planning is in de volgende paragraaf opgenomen.

3.3. Projectplanning

Aan de hand van de Mini KADS methodiek en de lijst met actiepunten (zie cyclus 1 verderop) is een planning opgesteld voor de projectperiode. Per cyclus is gespecificeerd welke werkzaamheden er uitgevoerd moeten worden, in welke periode en wat de eindresultaten van die cyclus zijn.

Projectidentificatie (cyclus 0)

De eerste cyclus levert:

- probleembeschrijving

Periode: week 1

Projectspecificatie (cyclus 1)

Tijdens deze cyclus wordt het probleem bestudeerd, zodat besloten kan worden of het zinvol is aan het project te beginnen. Er wordt gestart met het bestuderen van papierfabricage en functionaliteit van zetmeel in het wetend. Het domein wordt verkend, zodat er effectief met de experts gesproken kan worden en er een begin gemaakt kan worden met de domeinlaag van het expertisemodel. Vanuit de nogal globale probleembeschrijving wordt het probleem afgebakend en worden de projectdoelen vastgelegd. Verder moeten er verschillende alternatieve oplossingsmethoden bekeken worden, waarna er een afweging plaatsvindt. De eerste contacten met mensen buiten Vertis worden gelegd, zodat de eerste oriënterende gesprekken kunnen plaatsvinden. Kennisbronnen en hun beschikbaarheid worden gedefinieerd.

Het eindprodukt van cyclus 1 is een projectvoorstel waarin in ieder geval de volgende punten worden gespecificeerd:

- projectdoelen
- haalbaarheid
- taak-agent-model (voorlopige versie)
- communicatie-ontwerp-model (globale schets)
- kennisbronnen en hun status
- projectorganisatie
- projectplanning
- specificatie van eind- en tussenprodukten

Aan het einde van de cyclus vindt er een gesprek plaats met de betrokkenen waarin besloten wordt of het project doorgang kan vinden.

Periode: week 1 t/m week 6

Systeemspecificatie (cyclus 2)

Het communicatie-ontwerp-model (functioneel ontwerp) wordt eerst globaal uitgewerkt. Vervolgens vindt een werkverdeling plaats, gevolgd door een gedetailleerde uitwerking. Er wordt vastgelegd welke in- en uitvoer het systeem moet hebben en in welke vorm. Met de betrokken partijen (AVEBE Papierlab en begeleiders ProMod) vindt op gezette tijden overleg plaats omtrent de precieze functionaliteit van het systeem. Daarnaast wordt in samenwerking met de expert(s) d.m.v. een aantal interviews het expertisemodel gedefinieerd. Naast de domeinlaag, waarmee al in de vorige cyclus een begin gemaakt is, worden nu de inferentie- en taaklaag gedefinieerd. Tevens wordt de status van de bij het Papierlab aanwezige *database* onderzocht. Eerst wordt gekeken of deze database beschikbaar is. Als dit het geval is, moeten de structuur en de omvang bekeken worden. Ten slotte wordt de ontwikkelomgeving G2 uitgebreid bestudeerd, zodat de systeembouw in de volgende cyclus kan plaatsvinden.

Deze cyclus levert het volgende op:

- rapportage omtrent database van het Papierlab
- in detail uitgewerkt expertisemodel
- in detail uitgewerkt communicatie-ontwerp-model

Periode: week 7 t/m week 14

Systeembouw (cyclus 3)

In deze cyclus worden de resultaten van de vorige cycli omgezet in een prototype van het toekomstige systeem. Eerst zal een werkverdeling worden opgesteld, waarin de programmeertaken per persoon worden weergegeven. Er wordt veel aandacht aan de interface besteed, zodat ProMod de mogelijkheden van het uiteindelijke systeem kan demonstreren. Indien de database van het Papierlab beschikbaar is, kan aan de hand van een aantal praktijkgevallen de werking van het systeem getoetst worden.

Deze cyclus levert op:

- programma versie nul (eerste prototype)

Periode: week 15 t/m week 24

"Gebruik(er)stest" (cyclus 4)

De bevindingen en resultaten uit de vorige cycli worden verwerkt in een eindverslag. Dit verslag wordt continu bijgewerkt gedurende alle cycli. Tijdens dit project zal het eindresultaat van deze cyclus grotendeels hetzelfde

zijn als dat van de vorige cyclus. Omdat de projectdoelstelling het afleveren van een prototype is, is er geen sprake van een definitieve versie. Het afgeleverde programma van dit project zal nog verder ontwikkeld worden binnen Vertis.

Deze laatste cyclus levert twee produkten op, te weten:

- eindverslag
- definitief programma (binnen dit p roject)

Periode: week 25 t/m week 26

Evaluatie

Het is niet gelukt om de planning aan te houden. De eerste twee cycli leverden geen problemen op, maar bij het tot in detail uitwerken van het communicatie-ontwerpmodel (functioneel ontwerp) liepen we achterstand op. Dit was te wijten aan een moeizame totstandkoming van de specificatie van de datastructuur. Dit was ook de periode waarin de opdracht is aangepast, omdat de uitgangspunten niet goed waren. De werkverdeling en het prototype van het programma werden daardoor later afgerond dan gepland. De medewerkers van het Papierlab hebben het systeem tussentijds niet getest (de "gebruikerstest"), maar alleen van ons te horen gekregen hoe het zou functioneren en (met schermafdrukken) hoe het eruitzag. Een tweede factor waardoor het project vertraging opgelopen heeft, was de inwerkperiode in G2. Het systeem kent een aantal eigenaardigheden waar we tijdens de implementatie tegenaan liepen. Uiteindelijk is de gehele afstudeerperiode bijna twee maanden uitgelopen en kan worden geconcludeerd dat deze planning niet goed is opgezet. Het niet rond kunnen krijgen van een goede datastructuur heeft de meeste tijd gekost.

4. PAPIERFABRICAGE

Dit hoofdstuk beschrijft het papierfabricageproces en geeft dus achtergrondinformatie voor het project. Deze informatie vormt de basis voor de kennis van het systeem. Bij de fabricage van papier worden diverse hulpstoffen gebruikt om het proces te verbeteren of de kwaliteit van het papier te beïnvloeden. Deze hulpstoffen of additieven komen in paragraaf 4.2 aan bod. Eén van de belangrijkste additieven is zetmeel. Zetmeel wordt onder andere gebruikt om de sterkte van het papier te verbeteren. In paragraaf 4.3 wordt uitgelegd wat zetmeel is en komt de *derivatisering* van zetmeel aan de orde. Verder wordt besproken welke functies het zetmeel in het papier heeft. Voor een uitgebreide beschrijving van het papierfabricageproces wordt de lezer verwezen naar [Smook, 92]. In de laatste paragraaf worden de belangrijkste *proceseigenschappen* beschreven. Voornamelijk op basis van deze eigenschappen trekt de menselijke expert conclusies over het papierfabricageproces. Dit is belangrijke kennis voor toepassing in SIX.

4.1. Fabricageproces

De belangrijkste grondstof in papier is cellulose. Cellulosevezels komen voor in onder andere vlas, stro, hout en katoen. Hout is veruit de belangrijkste bron voor cellulosevezels voor de papierfabricage. De vezels kunnen op chemische of mechanische wijze vrij gemakkelijk uit het hout gewonnen worden. Allereerst wordt het hout versnipperd en vermengd met water. Wanneer de vezels op chemische wijze gewonnen worden, dan wordt het hout gezamenlijk met een aantal chemicaliën gekookt totdat de vezels makkelijk van elkaar kunnen worden gescheiden. Bij de mechanische methode wordt de houtpulp door een schijvenmolen van tegen elkaar wrijvende kollerstenen of een pulper met draaiende rotoren geleid. Hierdoor worden de vezels losgeslagen en gesplitst. De aldus ontstane vervezelde brij wordt hierna verder gemalen in een refiner. De gemalen vezels zijn korter en de celwanden zijn enigszins gekneusd en daardoor soepeler. Doordat de vezels water hebben aangetrokken, zijn ze opgezwollen. In dit stadium van het proces worden de zogenaamde *natpartij*-toevoegingen aan het proces toegevoegd. Vervolgens wordt de papierpulp verdund, schoongemaakt en naar de eigenlijke papiermachine gevoerd. De verdunde papiersuspensie komt aan in de *oploopkast* en wordt gelijkmatig over de breedte van een lange zeef verdeeld. Op deze zeef vindt de eigenlijke vorming van papier plaats. Het merendeel van het water valt door de zeef en door het ontstane blad tussen walsen uit te persen wordt het ontdaan van het resterende water. Het deel van het proces waar de natpartij-toevoegingen worden toegevoegd (tot aan de persen) wordt het wet-end deel van de papierfabricage genoemd. Ten slotte wordt door verdamping het vochtgehalte teruggebracht tot de gewenste waarde in de droogpartij.

Veredeling van het papier vindt plaats in de lijmpers . De banen papier worden door een oplossing van hulpstoffen gevoerd. Deze stoffen worden vervolgens door rollen in het papier geperst. Door deze procedure worden bijvoorbeeld de beschrijfbaarheid en bedrukbaarheid van het papier verbeterd. Nadat het papier verder gedroogd is, krijgen bepaalde papiersoorten nog een deklaagje. Deze procedure wordt het strijken of coaten van het papier genoemd. Het vergroot de oppervlaktesterkte waardoor het papier beter bedrukbaar wordt. Verder dient het vooral ter verfraaiing van het uiterlijk van het papier (witheid, ondoorschijnendheid en glans). Het coaten kan zowel aan één zijde als aan twee zijden van het papier gebeuren. Dit laatste wordt wel gedaan om de tweezijdigheid van het papier te verminderen. Omdat bij de produktie één kant van het papier op de zeef ligt, verschilt deze in gladheid en structuur van de andere kant. Dit wordt tweezi jdigheid genoemd. De laatste stap bij de produktie is het op maat snijden van het papier.

4.2. Additieven

Op verschillende punten in het fabricageproces worden hulpstoffen (additieven) toegevoegd om bepaalde papiereigenschappen te bereiken. Na toevoeging wordt er over het algemeen een bepaalde fysische of chemische binding gevormd tussen de vezel en de hulpstof. Tevens worden de hulpstoffen gebruikt om bepaalde proces-omstandigheden te wijzigen waardoor het fabricageproces geoptimaliseerd wordt en het papier een betere kwaliteit krijgt. Tabel 4.1. is een overzicht van de soorten hulpstoffen die gebruikt worden bij de papierproduktie [van Bergen, 86]. In de rest van deze paragraaf worden de belangrijkste nog uitgebreider beschreven.

Soort hulpstof	Voorbeelden	Voornaamste functie
Vulstof	Porseleinaarde, krijt	Verhoging opaciteit en bedrukbaarheid van papier
Lijmstof	Harslijm	Vermindering water-absorptie
Bindmiddel	Zetmeel, CMC	Verhoging papiersterkte
Natsterktemiddel	Ureum-formaldehydehars	Verhoging natsterkte papier
Retentiemiddel	Zetmeel, synthetische polymeren	Verhoging retentie vulstof op zeef
Fixeermiddel	Aluin, diverse polymeren	Hechting van verschillende hulpstoffen aan de vezel
Kleurstof	Pigmenten en kleurstoffen	Bijtinten of kleuren
Opwitter	Stilbeenderivaten	Verhoging witheid
Ontwateringsversneller	Synthetische polymeren	Snelle ontwatering op de zeef en snelle droging
pH-regelaar	Aluminaat	Wijzigen van de zuurgraad in het proces
Anti-slijmmiddel	Chloorverbindingen, enzymen	Tegengaan van bacteriegroei in de papierstof
Anti-schuimmiddel	Alcoholderivaten, siliconen	Tegengaan van schuim in stofvoorbereiding en papier

Bron: [van Bergen, 86]

Tabel 4.1. Enkele belangrijke hulpstoffen

Het eerste type hulpstof dat gebruikt wordt, is vulstof (filler). Vulstoffen maken het papier wat slapper en daardoor geruislozer. Geruisloosheid van papier kan bijvoorbeeld wenselijk zijn voor programmaboekjes. Verder kunnen vulmiddelen de drukeigenschappen verbeteren, de ondoorlaatbaarheid van licht (opaciteit) verhogen en het oppervlak witter en gladder maken. Veel gebruikte vulstoffen zijn porseleinaarde, gips, titaanwit, talk en krijt. Een heel belangrijke reden voor het gebruik van vulstoffen is de relatief lage prijs. Als het mogelijk is om meer vulstoffen toe te voegen zonder dat dit invloed heeft op het papier (eventueel doordat andere middelen de sterkte weer verbeteren) is dat meestal vanwege de prijs voor de producent interessant.

Een andere groep hulpstoffen wordt gevormd door de lijmstoffen. Hieronder vallen onder andere harslijmen, was-emulsies en synthetische lijmmiddelen. Omdat de vezels en de vulstoffen in het papier vochtabsorberend zijn moet het papier behandeld worden om het goed beschrijfbaar te maken. Lijming van het papier vergroot de waterafstotendheid en maakt het papier beter beschrijfbaar. Een ander belangrijk effect van lijmstoffen is het vergroten van de papiersterkte. Lijmstoffen kunnen zowel in de natpartij (inwendige sterkte) als tijdens de oppervlaktelijming (oppervlaktesterkte) in de lijmpers worden ingezet.

Andere hulpstoffen die de sterkte verhogen zijn bindmiddelen. Opgelost in water worden zetmeelstijfsels veel toegepast als bindmiddel. Een ander bindmiddel dat veel wordt toegepast is de chemische stof CMC, ondanks het feit dat deze aanmerkelijk duurder is.

Natsterktemiddelen zorgen ervoor dat de natsterkte van het papier wordt verhoogd. Wanneer het papier nat wordt, zorgen deze middelen dat er minder verlies aan vezelbinding optreedt.

Retentiemiddelen zorgen ervoor dat tijdens de ontwatering op de zeef minder vezels en vulstoffen uit het papier spoelen. De retentiemiddelen zorgen ervoor dat de vezelbinding, die bij de vulstoffen vaak niet goed

genoeg is, groter wordt waardoor er meer hulpstoffen achterblijven in het papier. Als retentiemiddel worden vaak dierlijke lijmen, zetmeel (zie paragraaf 4.3.) of synthetische middelen gebruikt.

Fixeermiddelen zijn stoffen die tijdens de bladvorming de hulpstoffen aan de vezels hechten. Aluminiumsulfaat, het papiermakers-aluin, is het belangrijkste fixeermiddel. De werking hiervan is afhankelijk van de zuurgraad in het proces, de temperatuur en de lading van de vezels en de aanwezige hulpstoffen. Naast het aluin zijn er in de papierwereld tegenwoordig ook synthetische middelen in gebruik.

Van de overige in de tabel genoemde groepen stoffen zijn alleen de anti-slijmmiddelen van belang voor het wet-end. Op vezelresten, koolhydraten, eiwitten en zouten kunnen bacteriën en schimmels groeien. Deze bacteriën en schimmels vinden we terug als slijm dat vlekken en breuken in het uiteindelijke papier veroorzaakt. Vooral bij een gesloten watersysteem zijn veel voedingsstoffen aanwezig die in het systeem blijven "zweven". Door middel van bacterieverdelgende middelen en fungiciden wordt getracht de bacteriën te doden. Hiervoor wordt gebruik gemaakt van diverse chloor- en broomverbindingen en zwavelstoffen. Verder is een goede beluchting van belang. Tegenwoordig wordt ter bestrijding van het slijm steeds vaker gebruik gemaakt van enzymen die het slijm oplossen.

De hechting van de hulpstoffen in het papierblad (en daardoor de uiteindelijke eigenschappen van het papier) zijn afhankelijk van de procesomstandigheden in het wet-end. De concentraties van chemische stoffen in het proceswater spelen hierbij de grootste rol. Alvorens verder in te gaan op de procesomstandigheden in het wet-end wordt in de volgende paragraaf eerst uitgelegd welke rol zetmeel in het papier en het fabricageproces speelt.

4.3. Zetmeel

In de vorige paragraaf is geschetst hoe de verschillende hulpstoffen bij de fabricage van papier worden ingezet. Sommige hulpstoffen worden toegevoegd om de eigenschappen van het papier te beïnvloeden. Daarnaast wordt een aantal hulpstoffen ingezet om de procesomstandigheden te wijzigen. Deze paragraaf geeft een beschrijving van zetmeel, zetmeelderivaten en het gebruik van zetmeel in het papier. In 4.3.1. wordt dieper ingegaan op zetmeel en derivaten. Paragraaf 4.3.2. geeft uitleg over het gebruik van zetmeel ter verhoging van de interne sterkte. In 4.3.3. wordt zetmeel als retentiemiddel beschreven. Dit zijn de twee belangrijkste functies van het zetmeel in het wet-end. Tot slot wordt in de laatste subparagraaf 4.3.4. een korte opsomming gegeven van andere mogelijke functies van het zetmeel.

4.3.1. Zetmeel en zetmeelderivaten

Zetmelen zijn natuurlijke koolwaterstofpolymeren die samengesteld zijn uit glucose-eenheden. Zetmeel wordt gewonnen uit gewassen, die het zelf gebruiken als reservevoedselvoorraad voor perioden van rust, kiem en groei. In deze gewassen zijn zetmeelkorrels opgeslagen in de zaden, knollen of wortels of de kern van de stam. De voornaamste commerciële bronnen van zetmeel zijn de zaden van granen, zoals mais, tarwe, sorghum en rijst, de knollen van aardappelen, de wortels van tapioca en de kern van de sago palm.

Aardappelzetmeel wordt gewonnen uit fabrieksaardappelen. Er wordt onderscheid gemaakt tussen consumptieaardappelen, pootaardappelen en fabrieksaardappelen. Deze laatste hebben een hoger zetmeelgehalte dan de andere en worden louter geteeld voor verwerking tot aardappelzetmeel in een zetmeelfabriek. De produktie van zetmeel is een eenvoudig proces. Allereerst worden de aardappels gewassen en door een raspmachine vermalen. De gemalen aardappelen worden vervolgens door een serie draaiende zeven geleid. Hierdoor blijven de vezels achter, die zo afgescheiden kunnen worden als aardappelpulp. De resterende zetmeelmassa bevat nog opgeloste bestanddelen zoals suikers, eiwitten, zuren, zouten en fijne vezels. Door middel van centrifuges of hydrocyclonen en fijne zeven worden deze verder gescheiden. De nu gezuiverde zetmeelmassa wordt gebruikt voor de produktie van aardappelzetmeelderivaten of wordt ontdaan van water en gedroogd. Het ruwe zetmeel dat niet verder wordt bewerkt, wordt natief zetmeel genoemd.

Zetmeel is samengesteld uit kleine witte korrels die in grootte van diameter varieren van 1 tot 100 micrometer. De vorm en grootte van deze korrels en de eigenschappen van het zetmeel zijn afhankelijk van het soort gewas waaruit het zetmeel is onttrokken. De meeste zetmelen bevatten twee typen glucosepolymeren, die

in verschillende verhoudingen voorkomen in de diverse natuurlijke zetmeelbronnen. Het eerste type is amylose, een lineair ketenmolecuul, terwijl amylopectine, de tweede vorm, een vertakt en aanmerkelijk groter glucosepolymeer is.

Derivatisering

Van zetmeel kunnen verschillende derivaten gemaakt worden. Ieder van de varianten heeft specifieke eigenschappen die het in meer of mindere mate geschikt maken voor de verschillende functies van het zetmeel. De zetmeelderivaten, die ook wel gemodificeerde zetmelen worden genoemd, zijn zetmelen waarvan de polymeerstructuur gewijzigd is ten opzichte van natief zetmeel. Derivatisering is dus een proces waarbij de scheikundige samenstelling van zetmeelmoleculen gewijzigd wordt. Het modificeren kan op fysische, chemische of enzymatische wijze plaatsvinden. De structuur van de moleculen wordt fysisch gewijzigd door verhitting of extrusie onder afschuifkrachten. Daarnaast kan natief zetmeel een behandeling met zuur ondergaan of wordt het geoxydeerd, veresterd of veretherd. Andere manieren van chemische aanpassing zijn hydrolyse en crosslinking. Tenslotte is het mogelijk om het zetmeel bloot te stellen aan inwerking van enzymen. In tabel 4.2. zijn de belangrijkste eigenschappen van zetmeel opgenomen. Het doel van derivatisering is het verbeteren van deze eigenschappen.

Viscositeit
Ionisch karakter
Gedrag in water
Pasting-temperatuur
Gelatinisatiegedrag bij verhitting
Neiging tot retrogradatie (stabiliteit)
Weerstand tegen afbraak

Tabel 4.2. Eigenschappen van zetmeelderivaten

De structuurverandering die plaatsvindt in de zetmeelmoleculen kan betrekking hebben op de lengte van de polymeerketens. De polymeermoleculen van het natief zetmeel kunnen als het ware in stukjes geknipt worden. Vooral de viscositeit van het zetmeel wordt op deze wijze beïnvloed; kleinere ketens komen overeen met lagere viscositeiten. De structuurwijziging kan ook betrekking hebben op de lading van de moleculen: er kunnen bepaalde molecuulgroepen "vastgeplakt" worden aan zetmeelketens. Wanneer er positief geladen groepen bij de zetmeelketens aangebracht worden, spreekt men van kationisch zetmeel, terwijl negatief geladen groepen het zetmeel anionisch maken. De zetmeelmoleculen van natief zetmeel hebben een ionisch neutraal karakter. Een andere factor die beïnvloed wordt door derivatisering is de zogenaamde pasting– of gelatinisatie-temperatuur. Zetmeelkorrels zijn niet oplosbaar in water kouder dan 50 °C. Wanneer een zetmeelsuspensie in water wordt verwarmd boven een bepaalde kritieke temperatuur, de pasting-temperatuur, zwellen de korrels op door wateropname en worden zij vele malen groter dan zij oorspronkelijk waren. Bij verdere verhitting beginnen de korrels uiteen te vallen, wat resulteert in een stroperige oplossing. Deze eigenschap van zetmeel is een belangrijke reden voor het gebruik van zetmeel in vele toepassingen.

Ook retrogradatie is een kenmerk van zetmeelderivaten. Retrogradatie is een proces waarbij zetmeelmoleculen die opgelost zijn in een zetmeelmassa geleidelijk en onomkeerbaar samenklonteren. De zetmeelmassa dikt hierdoor in en de viscositeit stijgt. De oplossing verandert in een substantie waarin zich gelei-achtige delen bevinden die niet langer opgelost zijn in water. Retrogradatie wordt meestal gezien als een ongewenste eigenschap van het zetmeel. Met name de amylosemoleculen in het zetmeel veroorzaken de retrogradatie. De lineaire ketens van deze moleculen komen bij retrogradatie parallel aan elkaar te liggen waarna er dwarsverbindingen gevormd worden. Bij amylopectinemoleculen komt dit, door de vertakte molecuulstructuur, nauwelijks voor. Het natieve maiszetmeel, dat een relatief hoog percentage amylose bevat, vertoont dus veel eerder retrogradatie dan de andere natieve zetmeelsoorten. Een andere factor die de retrogradatie beïnvloedt is de lengte van de zetmeelmoleculen. Een korte molecuulketen versterkt de neiging tot retrogradatie. Er wordt ook wel gesproken over lage-viscositeitsretrogradatie; immers een korte keten komt overeen met lage viscositeit. Om retrogradatie tegen te gaan wordt zetmeel gestabiliseerd in een derivatiseringsproces.

De laatste belangrijke eigenschap waarop zetmeelderivaten beoordeeld worden is de weerstand tegen afbraak onder invloed van zuren, hitte en afschuifkrachten. Afbraak van zetmeelmoleculen in een substantie

veroorzaakt een verlaging van de viscositeit. In tabel 4.3. is een aantal klassen van modificaties vermeld, Tevens is het belangrijkste doel van de derivatisering en het soort behandeling weergegeven.

Type modificatie	*Voornaamste doel*	*Behandeling*
Voorgekookt zetmeel	Gedrag in koud water	Verhitting door drumdrying, extrusie
Lage-viscositeitszetmelen	Lagere viscositeit	
• Dextrines	Lagere viscositeit, stabiliteit	Verhitting met zuur
• Zuur-gemodificeerd zetmeel	Lagere viscositeit, verbeterde gelatinisatie	Hydrolyse onder zuur
• Geoxydeerd zetmeel	Lagere viscositeit, stabiliteit	Oxydatie
• Enzymatisch gemodificeerd zetmeel	Lagere viscositeit	Inwerking van enzymen
Crosslinked zetmeel	Gedrag bij verhitting	Crosslinking
Gestabiliseerd zetmeel	Stabiliteit, ionisch karakter	Verestering, verethering
Zetmeelsuikers	Zoete saccharide	Zuur of enzymen

Bron: [Swinkels, 91]

Tabel 4.3. Klassen van zetmeelderivaten

Het is ook mogelijk om modificaties te combineren. Door middel van verschillende behandelingen kan dan een combinatie van doelen bereikt worden. Zetmeelderivaten worden voor uiteenlopende toepassingen gebruikt. In de voedingsmiddelenindustrie worden ze verwerkt als zetmeelsuiker, bindmiddel, verdikkingsmiddel en stabilisator. Hierbij moet gedacht worden aan gebruik in vla, pudding, suikerwerken, bakkerijprodukten, soeppoeders, bliksoepen, etc. In de papierindustrie geeft zetmeel sterkte en klank aan het papier en het verbetert de beschrijfbaarheid. In de textielindustrie wordt zetmeel gebruikt om garen te sterken en ter verbetering van het bedrukken van stoffen. Verder wordt zetmeel gebruikt in de kleefstoffenindustrie, pharmaceutsche industrie en wordt het toegepast in veevoeder. In de volgende paragrafen zullen de functies van zetmeel in het papier toegelicht worden.

4.3.2. Zetmeel ter verhoging van de interne sterkte

In het wet-end deel van het papierfabricageproces wordt zetmeel onder andere gebruikt als bindmiddel om de sterkte-eigenschappen van het papier te verbeteren. Zetmeel kan de interne sterkte van papier, zoals trekssterkte, vouwweerstand en berstdruk, vergroten. De interne sterkte is een kenmerkende en essentiële eigenschap van een vel papier. Interne sterkte vloeit voort uit bindingen tussen de papiervezels, die ontstaan bij het drogen van het papierblad. De sterkte van het papier is dus afhankelijk van de sterkte van de afzonderlijke vezels in het blad en de sterkte van de verbindingen. Daarnaast zijn het aantal bindingen en de spreiding van de bindingen van invloed op de sterkte. Het toevoegen van bindmiddel kan voor een verbetering zorgen van de drie laatstgenoemde factoren.

De keuze van grondstoffen is de eerste factor die bepalend is voor de interne sterkte van papier. Langere cellulosevezels bijvoorbeeld geven papier een grotere sterkte dan korte vezels. De toevoeging van vulmiddelen zorgt voor een vermindering van de sterkte. Na de keuze voor grondstoffen kan de sterkte van het uiteindelijke papier nog vergroot worden door mechanische bewerking. Het papier wordt sterker door de behandeling die de pulp van cellulosevezels ondergaat in de refiner. Door de maling neemt de sterkte van de vezels iets af, maar de binding tussen de vezels wordt aanzienlijk verbeterd. Later in het proces is de sterkte nog enigszins te verbeteren door natte papierbanen te walsen. Ook procesomstandigheden zijn van invloed op de uiteindelijke sterkte. In een alkalische omgeving blijken vezels betere bindingen te vormen. Het gebruik van bindmiddelen is echter, naast de maling in de refiner, de belangrijkste manier om de interne sterkte te vergroten.

Zetmeel als bindmiddel vergroot zoals al gezegd de treksterkte, de vouwweerstand en de berstdruk. Daarnaast kan de scheurweerstand verbeterd worden bij een gelijkblijvende treksterkte wanneer de inzet van zetmeel gedeeltelijk het malen vervangt. Zetmeel als bindmiddel maakt het mogelijk om een hoger aandeel aan minder sterke vezels te gebruiken. Bovendien kan er meer vulmiddel ingezet worden zonder dat dit ten koste gaat van de sterkte. Het zetmeel creëert nieuwe bindingen tussen de cellulosevezels door waterstofbruggen te vormen of electrostatische ion-bindingen. Het lineaire amylosedeel van het zetmeel wordt met de waterstofbruggen parallel aan de cellulosevezels in het vezelblad geadsorbeerd. De vertakte amylopectinemoleculen raken verstrikt met de cellulosevezels en hechten op deze wijze met waterstofbruggen het papier. In de regel wordt in het wet-end kationisch zetmeel gebruikt als bindmiddel. Kationisch zetmeel vormt naast de waterstofbruggen ook ion-bindingen met de anionische cellulosevezels. Door deze sterke bindingen is de adsorptie van kationisch zetmeel vrijwel onomkeerbaar. Het zetmeel is het meest effectief wanneer er een juiste verdeling is tussen de van nature aanwezige anionische fosfaatgroepen en de later, tijdens derivatisering toegevoegde, kationische elementen.

Er is een aantal redenen waarom de retentie van zetmeel in het papierblad belangrijk is. Wanneer een groter gedeelte van het zetmeel dat wordt ingezet in het wet-end achterblijft in het papierblad, kan met een kleinere hoeveelheid zetmeel volstaan worden om hetzelfde zetmeelgehalte in het uiteindelijke papier te bereiken. Een hogere retentie betekent dat de efficiëntie van het zetmeel groter is aangezien er minder zetmeel ongebruikt achterblijft in het proceswater. Daarnaast kan het zetmeel dat niet retendeert het papierfabricageproces verstoren. In moderne papierfabrieken wordt het proceswater meerdere malen gebruikt alvorens het wordt gezuiverd of geloosd. Zetmeel dat is achtergebleven in het proceswater kan gaan werken als stoorstof wanneer het opnieuw in het wet-end terecht komt. Ten slotte is een hoge retentie wenselijk omdat zetmeel dat niet retendeert uiteindelijk uit het afvalwater gezuiverd moet worden. Wanneer er geen hulpmiddelen zoals papiermakers-aluin gebruikt worden, retendeert natief zetmeel voor niet meer dan ongeveer 30%. Kationisch zetmeel daarentegen hecht zich vrijwel volledig aan het papier.

4.3.3. Zetmeel als retentiemiddel

Een tweede belangrijke functie van het zetmeel in het wet-end is het verbeteren van de retentie van andere hulpmiddelen. Het maken van papier kan beschouwd worden als een filtratieproces. De zeef in de papiermachine is het filter waar een gedeelte van de stoffen die toegevoegd zijn aan de papierpulp achterblijven in het papierblad. De stoffen die niet achterblijven worden uitgezeefd en afgevoerd met het zogenaamde witwater. Deze stoffen komen opnieuw in het proces terecht of worden geloosd in het afvalwater. Bepaalde stoffen verhogen de retentie van vulmiddelen en andere hulpstoffen. Deze stoffen, retentiemiddelen, zorgen er tevens voor dat ook de kleine vezels beter retenderen en doordat de hulpstoffen sneller hechten aan het papier wordt het mogelijk de snelheid van de papiermachine op te voeren (waardoor de produktiviteit van de machine stijgt). Bovendien betekent een betere retentie dat er minder kostbare grondstoffen verloren gaan en de waterzuivering ontlast wordt.

Retentiemiddelen vormen veelal ionische bindingen tussen de vulmiddelen en andere hulpstoffen en de cellulosevezels. Aangezien papiervezels enerzijds en vulmiddelen en hulpstoffen anderzijds in het algemeen negatief geladen zijn, kunnen kationische stoffen dienen als retentiemiddel . De kationische retentiemiddelen hechten zich zowel aan de vezels als aan de hulpstoffen en vormen op deze manier verbindingen tussen gelijk geladen stoffen. Kationische hulpstoffen hechten zich ook zonder hulpmiddelen in het anionische papierblad. De effectiviteit van een retentiemiddel blijkt niet alleen afhankelijk van de ladingsdichtheid van het papierblad, de te hechten hulpstof en het retentiemiddel zelf, maar ook van de grootte en molaire massa van de vulstof en van de zuurgraad van de procesomgeving.

Kationisch zetmeel kan de retentie van hulpstoffen aanzienlijk verbeteren en wordt daarom ook onder de retentiemiddelen gerekend. Toch wordt dit meestal beschouwd als een, plezierig, neveneffect. Het vergroten van de interne sterkte is de belangrijkste reden voor het gebruik. De grootte van de kationische lading van het zetmeel is bepalend voor de effectiviteit van zetmeel als retentiemiddel.

Naast het verbeteren van de interne sterkte en retentie kan er om een aantal andere redenen zetmeel gebruikt worden in het papierfabricageproces. In het wet-end kan zetmeel nog ingezet worden voor de inwendige lijming. Hiermee wordt de waterafstotendheid van het papier vergroot. Ook worden in het wet-end speciale dialdehyde-derivaten gebruikt om de natsterkte van het papier te vergroten. Naast gebruik in het wet-end wordt zetmeel tenslotte ook toegepast in de lijmpers en tijdens het strijken (of coaten) van het papier.

4.4. Proceseigenschappen in het wet-end

Wanneer een klant van AVEBE advies vraagt over het papierfabricageproces, heeft AVEBE daarvoor bepaalde gegevens nodig. Vaak wordt een aantal monsters van verschillende plaatsen in het proces door de klant geleverd. Het komt ook voor dat AVEBE zelf monsters neemt bij de klant en ter plekke een deel van het advies uitbrengt. De experts van het Papierlab trekken dus conclusies aan de hand van meetwaarden van de pulpmonsters. Deze meetwaarden, of proceseigenschappen, zijn daarom belangrijke gegevens voor het SIX systeem. In deze paragraaf worden puntsgewijs de vier belangrijkste eigenschappen beschreven en wordt per eigenschap uitgelegd wat het belang ervan is voor het bepalen van het advies.

pH

De pH in het systeem wordt door middel van een pH-meter bepaald. De grootheid pH geeft de zuurgraad van een oplossing of suspensie aan. De pH heeft een bereik van 0 tot 14. Wanneer de pH 7 is, is de oplossing neutraal. Lagere waarden geven aan dat de oplossing zuur is; hogere waarden dat de oplossing basisch is.

Uit de waarde van de pH kan de expert een aantal dingen afleiden. Wanneer de pH neutraal is, is het mogelijk dat het proces geschikt is voor het gebruik van kationisch zetmeel. Wanneer de pH zuur is, wordt er waarschijnlijk alum gebruikt in het proces (het proces is al kationisch). In hoeverre het nog mogelijk is kationisch zetmeel te gebruiken is afhankelijk van de hoeveelheid alum. Wanneer er te veel alum toegevoegd is en het niet mogelijk is deze dosering te verlagen moet er overwogen worden anionisch zetmeel te gebruiken. De pH is van invloed op de zetapotentiaal en de op effectiviteit van chemische additieven.

Specific Conductivity (Sc)

De Sc van de pulp is afhankelijk van de hoeveelheid geioniseerd materiaal in de pulp. In het wet-end betekent dit dat de Sc voornamelijk afhankelijk is van de hoeveelheid zout die aanwezig is in de pulp. Zout schermt geladen deeltjes als het ware af waardoor de effectiviteit van die deeltjes vermindert. Omdat ook kationische deeltjes door het zout afgeschermd worden, is de effectiviteit van een kationisch zetmeel afhankelijk van de Sc. Hoe meer kationische groepen een zetmeel bevat (of hoe hoger de degree of substitution van een zetmeel), hoe beter de afweer van het zetmeel is tegen het zout.

AVEBE gebruikt de Sc als indicatie voor het te gebruiken zetmeel, waarbij ze grofweg de volgende indelingen hanteert:

Sc:	=< 1000	- d.s.: 0.017
Sc:	1000-2000	- d.s.: 0.035
Sc:	>= 2000	- d.s.: 0.047

In extreme gevallen met een Sc groter dan 4000 is alleen een zetmeel met een nog hogere d.s. bruikbaar. De effectiviteit van andere geladen chemicaliën wordt op soortgelijke wijze als die van het zetmeel beïnvloed.

Zetapotentiaal (Zp)

De zetapotentiaal is de electrostatische lading van een deeltje in water. Het deeltje is in het water direct omringd door een laag vloeistof die ten opzichte van het deeltje niet beweegt. Dit is de zogenaamde "fixed layer". De potentiaal tussen deze fixed layer en de vloeistof wordt zetapotentiaal genoemd. Deze waarde geeft de afwerende kracht en tussen de deeltjes weer.

AVEBE hanteert als regel voor optimale retentie in het proces dat de Zp zo dicht mogelijk de waarde nul moet benaderen. De zetapotentiaal kan gewijzigd worden door het toevoegen van kationisch materiaal of door de pH te wijzigen. De zetapotentiaal wordt door AVEBE tevens gebruikt als indicatie van de hoeveelheid zetmeel die toegevoegd kan worden aan het proces.

Cationic/Anionic Demand (CD/AD)

Cationic en anionic demand zorgen voor de totale lading van de opgeloste stoffen in het proces. Cationic demand wordt veroorzaakt door opgelost anionisch materiaal, het zogenaamde anionic trash (AT) of anionische stoorstoffen. Anionic demand wordt veroorzaakt door kationisch materiaal ofwel cationic trash (CT). De naam stoorstof benadrukt meteen het belangrijkste effect van AT/CT in een proces. Het verstoort de werking van geladen chemicaliën. Op een vergelijkbare manier als beschreven voor het zout wordt de werking van de

chemicaliën verstoord door de trash. Afhankelijk van de hoeveelheid trash (waarvoor de CD/AD een indicatie is) besluit AVEBE of er een zogenaamde trash-catcher gebruikt moet worden alvorens zetmeel en andere additieven toe te voegen. Simpel gezegd reageert deze trash-catcher met de stoorstof alvorens de andere additieven daar de kans toe krijgen zodat de eigenlijke functionaliteit van die stoffen niet verloren gaat.

5. PROBLEEMCONTEXT

R.A. Carton: § 5.1., § 5.2.2.; M. Pruis: § 5.2.1., § 5.2.3.

SIX is een systeem dat experts advies geeft voor verbetering van het wet-end van het papierfabricageproces. Het is een demonstratieprogramma dat aangeeft hoe een geavanceerd, eventueel *real-time* sturend expertsysteem voor een papiermachine eruit zou kunnen zien en hoe het bediend zou kunnen worden. De eerste paragraaf van dit hoofdstuk geeft een generieke beschrijving van enerzijds het expertiseprobleem van SIX en anderzijds het interface-probleem. Om een indruk van de probleemcontext te krijgen, behandelt de tweede paragraaf (5.2.) vervolgens enkele systemen waarvan het expertiseprobleem op abstract niveau vergelijkbaar is met dat van SIX.

5.1. Abstractie van het probleem

Het probleem dat in hoofdstuk 2 is beschreven, bestaat uit twee delen. Het geven van advies is de ene kant van het probleem en het zorgen voor een eenvoudige bediening is de andere kant. Op beide zaken wordt hierondèr dieper ingegaan.

5.1.1. Expertgedeelte

Het expertprobleem waarvoor SIX is ontworpen kan in tweeën worden gedeeld: een analyserend deel en een adviserend deel. Deze delen worden eerst kort toegelicht en dan geabstraheerd.

Als de wet-end-gegevens van een zeker papierfabricageproces (dus een aantal pulp samples en gebruikte additieven) ter optimalisatie worden voorgelegd aan het Papierlab van AVEBE, zal men als volgt te werk gaan. Allereerst worden de gegevens van de pulp samples geclassificeerd om mogelijke probleemwaarden te onderkennen. Classificeren behelst het afbeelden van de meetgegevens. Voorbeelden hiervan zijn: "zeer hoog", "licht negatief" en "sterk basisch". Probleemwaarden zijn procesgegevens die niet binnen het interval liggen dat door één of meer regels wordt voorgeschreven. Een voorbeeld van zo'n probleemwaarde is de classificatie "zuur" van een gemeten pH in een situatie waar de pH alleen "neutraal" mag zijn. De regels zijn voornamelijk algemene vuistregels, maar voor een aantal gevallen kunnen de intervalgrenzen duidelijk worden aangegeven. De probleemwaarden hebben altijd één of meer oorzaken en het vinden van deze oorzaken is de volgende stap die door de experts wordt gedaan. Het vinden van een oorzaak is het laatste onderdeel van de diagnosefase.

Als er oorzaken gevonden zijn, is de oplossing ook bekend. In het algemeen zal een afwijking het gevolg zijn van het in verkeerde volgorde doseren van additieven, of van het doseren van te kleine of te grote hoeveelheden van een additief. Het advies van AVEBE aan de betreffende papierproducent zal dan ook zijn om de volgorde van dosering te veranderen, dan wel de doses aan te passen. De experts zullen, indien dit mogelijk en wenselijk is, hun bevindingen eerst in een proefopstelling bij AVEBE testen en vervolgens tot een eindoordeel komen. Zij brengen dan advies aan de fabrikant uit en zullen na enige tijd vernemen of de voorgestelde wijzigingen tot verbetering van het proces hebben geleid. Hier eindigt de adviesfase van het proces.

De taak kan dus worden gekarakteriseerd als een diagnoseprobleem. Er zijn *off-line* procesgegevens die bestaan uit gemeten waarden en vooraf bekende parameters. De gegevens moeten als geheel worden gecontroleerd op afwijkingen van standaardwaarden die min of meer formeel in regels zijn vastgelegd. Voor de geconstateerde afwijkingen moeten oorzaken worden afgeleid. Uitgaande van de procesparameters en algemene

kennis van het proces kunnen daarvoor hypotheses worden opgesteld. De diagnose zal dus bestaan uit één of meer geconstateerde problemen in het proces, die kunnen worden uitgedrukt in verkeerd ingestelde procesparameters. Aan de hand van de gestelde diagnose kan wo rden geadviseerd de procesparameters te wijzigen.

5.1.2. Interface-gedeelte

SIX is bedoeld als demonstratiesysteem. Enerzijds moet het een indruk geven van de werking van een e xpertsysteem op het gebied van papierproduktie en anderzijds moet het eenvoudig te bedienen zijn. Deze laatste eis is belangrijk omdat de doelgroep (in de eerste plaats de medewerkers van het Papierlab van AVEBE) niet noodzakelijkerwijs kennis van computers heeft. Voor deze mensen moet SIX dus niet alleen makkelijk te bedienen zijn, maar ook een zekere aantrekkingskracht uitoefenen. Als het programma op het eerste gezicht niet interessant is of niet makkelijk in het gebruik, zal de doelgroep haar belangstelling snel verliezen.

Het probleem dat achter de interface voor SIX schuilgaat, is dus een presentatieprobleem met een esthetisch en een toegankelijkheidsaspect. Het belangrijk dat de interface een goede indruk op de gebruiker maakt. Het esthetische aspect van het probleem behelst de aantrekkingskracht van de interface op de gebruiker. Het programma moet uitnodigen om ermee te werken doordat het visueel aantrekkelijk is. Het andere aspect, de toegankelijkheid, kan worden bewerkstelligd door de bediening van het programma eenvoudig en begrijpelijk te houden. Als de gebruiker makkelijk met een programma kan werken en zich er iets bij kan voorstellen, is het toegankelijk. Deze twee zaken vormen het probleem van de interface, die het uiterlijk van het programma vormt en dus voor de presentatie zorgt. Het kiezen van een geschikte interface-vorm wordt in hoofdstuk 8 behandeld.

5.2. Vergelijkbare expertsystemen

In deze paragraaf worden enkele expertsystemen bekeken, waarvan het onderliggende probleem vergelijkbaar is met dat van SIX. Deze systemen hebben namelijk ook een diagnostische en een adviserende functie. Eerst wordt MYCIN beschreven. MYCIN is een medisch expertsysteem (eind jaren zeventig) dat voor veel systemen in dit vakgebied als voorbeeld heeft gefungeerd en nog steeds het klassieke voorbeeld is van een expertsysteem. Daarna wordt een ander medisch programma beschreven, dat SPHINX heet. Het laatst besproken systeem is EPAK. Het wordt hier besproken omdat het probleemgebied van EPAK voor een groot deel overeenkomt met onze probleemstelling en omdat het in G2 is ontwikkeld.

5.2.1. MYCIN

Het MYCIN project was een gezamenlijk project van de informatica en de medische faculteit van Stanford University. Het belangrijkste werk van dit project vond plaats tijdens het eind van de zeventiger jaren en er zijn ongeveer 50 manjaren aan het project besteed. Als één van de eerste ontworpen en gerealiseerde expertsystemen heeft het een belangrijke rol gespeeld bij de ontwikkeling het vakgebied [Luger, 89]. Eén van de redenen waarom het zo invloedrijk geweest is, is de uitgebreide projectdocumentatie die door de verschillende Stanford research teams is samengesteld [Buchanan, 84]. MYCIN werd gerealiseerd in INTERLISP, een LISP dialect.

MYCIN probeert de juiste diagnoses te stellen voor patiënten met bacteriële infecties. Er zijn tests om de oorzaak van een bacteriële infectie exact te bepalen maar deze geven pas uitsluitsel na een periode van 48 uur. In sommige gevallen kan snellere behandeling van de patiënt noodzakelijk zijn vanwege de pijn die de patiënt door de infectie heeft. Artsen hebben inmiddels een aanzienlijke ervaring opgebouwd, gebaseerd op eerste symptomen en testresultaten waarmee ze een diagnose kunnen stellen. Hiermee stellen ze ongeveer vast wat de oorzaak van de infectie is zodat de behandeling kan beginnen. Deze behandeling wordt dan als de testresultaten bekend zijn verder verfijnd. De ontwerpers van MYCIN hebben juist dit domein gekozen omdat het een voorbeeld is van hoe mensen redeneren en problemen oplossen op basis van incomplete of onbetrouwbare informatie [Luger, 89]. MYCIN was één van de eerste systemen waarbij gebruik werd gemaakt van zekerheidsfactoren (*certainty factors*) in een op regels gebaseerd expertsysteem [Rich, 91].

Feiten worden in MYCIN gerepresenteerd door middel van een attribuut-object-waarde koppeling. Het eerste element in deze structuur is een eigenschap/attribuut van een bepaald object uit het domein. Voorbeelden van deze eigenschappen zijn bijvoorbeeld de identiteit van een bacterie/organisme of de gevoeligheid voor penicilline van een organisme. Gekoppeld hieraan worden de naam van het object en de waarde opgenomen. Aan deze gegevens wordt verder nog een zekerheidsfactor gekoppeld waarmee uitgedrukt wordt hoe (on)zeker de waarde van het feit is. Een uitgebreidere beschrijving van de basis van deze zekerheidsfactor vindt de lezer in paragraaf 7.4.1. In LISP statements zou het feit dat een organisme niet gevoelig is voor penicilline er als volgt uitzien:

```
(gevoeligheid organisme_1 penicilline -1.0)
```

De zekerheidsfactor kan een waarde aannemen van -1 tot en met 1. Hierbij geeft de waarde 1 absolute zekerheid aan dat het feit waar is en -1 absolute zekerheid dat iets niet waar is. De waarde 0 geeft aan dat er niets bekend is over een feit [Luger, 89]. Er kan in MYCIN dus ook geredeneerd worden met feiten wanneer bekend is dat ze (met een bepaalde zekerheidsfactor) niet waar zijn.

Een regel in het MYCIN systeem zou er bijvoorbeeld zo uitzien [Rich, 91]:

```
If: (1) the stain of the organism is gram-positive, and
    (2) the morphology of the organism is coccus, and
    (3) the growth conformation of the organism is clumps,
then there is suggestive evidence (0.7) that the identity of the organism is staphylococcus.
```

Dit is de vorm waarin de kennis gerepresenteerd (en gevraagd) wordt aan de gebruiker. Intern slaat het systeem de regel op in een LISP list-structuur:

```
PREMISE: ($AND  (SAME CNTXT GRAM GRAMPOS)
                (SAME CNTXT MORPH COCCUS)
                (SAME CNTXT CONFORM CLUMPS) )
ACTION:  (CONCLUDE CNTXT IDENT STAPHYLOCOCCUS TALLY 0.7)
```

5.2.2. SPHINX

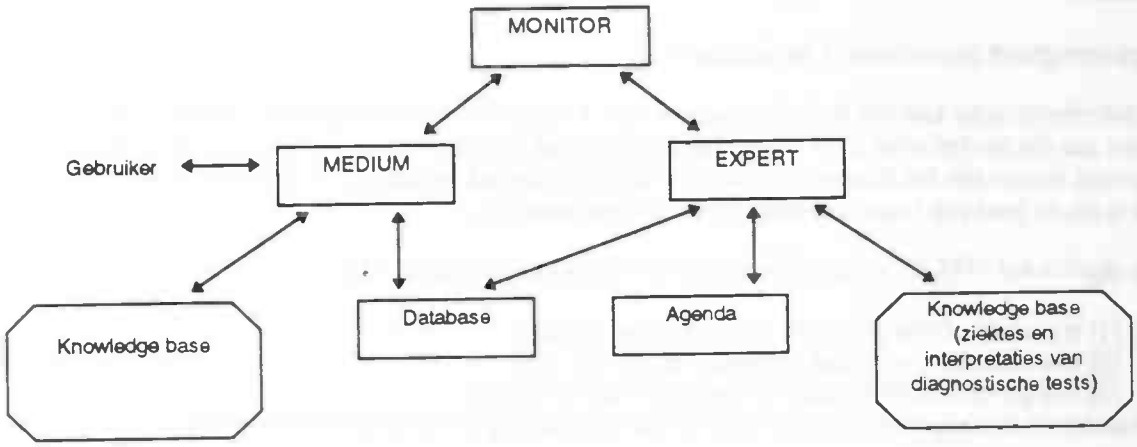
SPHINX is een systeem dat clinici moet bijstaan in het maken van beslissingen op diagnostisch, therapeutisch en preventief gebied [Fieschi, 90]. Er zijn vier doelen voor dit systeem opgesteld, te weten:

1. De kennis in het systeem moet zorgen voor advies dat vergelijkbaar is met dat van een expert.
2. De communicatie met het programma moet in een duidelijke taal geschieden, die zo dicht mogelijk bij de *natuurlijke taal* ligt, zodat de gebruiker geen taal of codering hoeft aan te leren.
3. Het systeem zou zijn beslissingen moeten kunnen verklaren door te laten zien waarop redeneringen zijn gebaseerd.
4. Het systeem zou moeten worden uitgerust met een data-acquisitiemodule die door een leek bediend kan worden. Hierdoor kunnen de adviezen worden verbeterd en veranderingen in medische kennis worden verwerkt. Bovendien zou het systeem breder inzetbaar worden.

Er is gebleken dat het voordelig is om een medisch expertsysteem de rol van dokter te geven, om op deze manier informatie over de patiënt te verkrijgen. Bij een gesprek met de dokter geeft de patiënt soms spontaan informatie, waar de dokter niet om heeft gevraagd. Deze informatie kan echter belangrijk zijn bij het opstellen van een diagnose en daarom zou een expertsysteem ook in staat moeten zijn om dit soort extra gegevens te verwerken en een rol te laten spelen bij latere vragen voor de patiënt.

De algemene opzet voor SPHINX is weergegeven in figuur 5.1. De gebruiker vult de database van het systeem met patiëntgegevens via de module MEDIUM, die zorgt voor een consistente dialoog met de gebruiker door middel van de *knowledge base*. MEDIUM is via de module MONITOR (die een controlerende functie heeft) aan de module EXPERT gekoppeld. EXPERT voert de taken op de agenda uit, waarbij de gegevens uit de database en uit de knowledge base met kennis over ziektes en interpretaties van diagnostische tests worden gebruikt. EXPERT zorgt voor een relevante dialoog met betrekking tot het zoeken en vragen naar symptomen. Door de modulaire opzet is de communicatie met de gebruiker op verschillende niveaus correct. De knowledge

base is zelfs opgedeeld om te zorgen dat enerzijds consistente data wordt ingevoerd en anderzijds de specifiek medische kennis wordt gebruikt om een diagnose op te stellen. Als de gebruiker bijvoorbeeld invoert dat de patiënt koorts heeft, zal de module MEDIUM controleren of er niet eerder is opgegeven dat de temperatuur van de patiënt normaal is. Bovendien zal de module op eigen initiatief vragen over de koorts stellen, om er een beter beeld van te krijgen. De EXPERT-module kan dan gaan zoeken naar een ziekte met koortsverschijnselen die aan de gegevens van MEDIUM voldoen.



Figuur 5.1. De algemene structuur van SPHINX

Voor SPHINX wordt van entiteiten gebruik gemaakt. Entiteiten zijn bijvoorbeeld symptomen, clinische signalen, onderzoeksresultaten, diagnoses en therapieën. Elke entiteit heeft bepaalde beschrijvende attributen. Zo kunnen de attributen voor "pijn" zijn: de plaats, de intensiteit, de "hoeveelheid" en de omstandigheden die de pijn veroorzaken. Voor de waarden van de attributen wordt gebruik gemaakt van "semantische categorieën". Een eniteit zoals "trek" (honger) kan worden geklassificeerd door "verhoogd", "normaal", "verlaagd", "sterk verhoogd", "sterk verlaagd", "licht verhoogd" en "licht verlaagd". Na de omzetting van gegevens in kwalitatieve benamingen kan de EXPERT-module puur met symbolen gaan redeneren.

De expertkennis van het systeem is voornamelijk vastgelegd in *produktieregels*. Deze hebben de volgende vorm:

IF premissen THEN effecten

Elke premisse is een combinatie van entiteiten, die samen voor de effecten in de produktieregels kunnen zorgen. Zo'n effect kan bijvoorbeeld een uitbreiding van de database of de agenda zijn. Bij de aanvang van het redeneerproces worden er op de agenda enkele doelen gezet. Het systeem probeert alle doelen op de agenda te verwezenlijken, voor zover dit mogelijk is. Omdat de agenda gewijzigd kan worden, heeft de gebruiker via de dialoog invloed op het redeneerproces. Zo wordt dus de gang van zaken bij een gesprek tussen een patiënt en een menselijke dokter nagebootst.

5.2.3. EPAK

EPAK is een *on-line* real-time expertsysteem dat de operators in een papierfabriek ondersteunt en adviseert. Het systeem is in 1988 ontwikkeld door Norske Skogindustrier A/S in Noorwegen [Yeager, 90]. De fabriek van dit bedrijf waar krantenpapier geproduceerd wordt is de grootste papierfabriek in Noorwegen. De beschrijving in deze paragraaf is een samenvatting van een artikel van Per-Olav Opdahl: "EPAK, an expert system for the support of paper quality control" [Opdahl]. De functie van het systeem is het geven van advies omtrent het zo effectief mogelijk produceren van papier volgens bepaalde kwalitatieve specificaties. De belangrijkste taken van het systeem zijn:

- Papierkwaliteit beoordelen t.o.v. bepaalde papierspecificaties.
- Wijzigingen van procesparameters adviseren om afwijkingen van papierspecificaties te corrigeren.
- Effecten van de geadviseerde wijzigingen simuleren.

Organisatie van de kennis binnen EPAK

Om deze taken te kunnen uitvoeren is de kennis die in het systeem opgenomen is onderverdeeld in drie categorieën: beoordelingskennis, correctiekennis en proceskennis.

De beoordelingskennis wordt gebruikt voor de vergelijking van de gemeten waarden en de *klantspecificaties* met als resultaat een classificatie van de kwaliteit. Deze kennis komt uit bepaalde vuistregels betreffende het papierproduktieproces en de specificaties van de klant. De kwaliteit wordt vastgesteld door de gemeten waarde te vergelijken met een gespecificeerd toegestaan bereik. De menselijke factor van overweging wordt geïmiteerd door beschrijvingen te gebruiken zoals hoog, o.k., laag, te hoog en te laag. Bij deze beoordeling wordt tevens rekening gehouden met bepaalde optimale procesparameters.

De correctiekennis wordt gebruikt om bepaalde corrigerende acties in het papierfabricageproces te kunnen adviseren op basis van de kwaliteitsclassificaties. Zowel de effecten van (gecombineerde) acties op enkele eigenschappen en de effecten op combinaties van eigenschappen zijn gedefinieerd om ongewenste bijeffecten te voorkomen. De koppeling tussen de acties en de kwaliteitseigenschappen wordt gespecificeerd als een verandering in die eigenschappen. Bijvoorbeeld "treksterkte omhoog" of "doorzichtigheid omlaag en gladheid omhoog". Via prioriteiten die het systeem toekent aan de verschillende acties en combinaties daarvan en de beperkingen van de procesparameters probeert EPAK de mogelijke acties. In die gevallen waar aan de hand van exacte gegevens een optimale procesparameter bepaald kan worden bevat de correctiekennis ook de berekening om dit te doen. In de andere gevallen is er voor elke classificatie een vooraf gedefinieerde wijziging in procesparameters opgenomen.

De proceskennis specificeert de eerder ervaren effecten op de kwaliteit van het papier naar aanleiding van de corrigerende acties. De kennis is vastgelegd door middel van coëfficiënten in een zogenaamde kwaliteitsmatrix. De coëfficiënten zijn gekozen aan de hand van de ervaring van de operators. Deze matrix wordt gebruikt omdat er geen compleet mathematisch model beschikbaar is van de papiermachine of het proces. Proceskennis die niet in de matrix kan worden vastgelegd, zoals fysieke beperkingen van bijvoorbeeld de machine-instellingen, wiskundige functies en tijdafhankelijk gedrag, is opgeslagen in regels en functies. De proceskennis wordt samen met de correctiekennis gebruikt om te bepalen hoe het proces eruit ziet op het punt waar het advies gegeven moet worden en om de effecten van het advies te simuleren.

Implementatie in G2

EPAK werd voor het eerst als prototype geïmplementeerd met behulp van de M1 PC-Shell. Dit is een regel-gebaseerde shell die gebruik maakt van *pattern matching* om te redeneren. Omdat deze shell een aantal problemen opleverde bij het representeren van gecombineerde acties bij elke eigenschap en de omvang al gauw boven de 640K standaardgeheugen van de PC uitkwam werd voor de definitieve versie besloten geen gebruik te maken van de PC.

Uit een aantal verschillende tools viel uiteindelijk de keuze op G2 vanwege de aanwezige mechanismen om met tijd en tijdafhankelijkheden te redeneren en de uitgebreide mogelijkheden voor kennisrepresentatie. Daarnaast is G2 beschikbaar op verschillende platformen zodat het systeem geïnstalleerd kon worden in de verschillende al bestaande omgevingen. Op de ontwikkelomgeving G2 wordt uitgebreider ingegaan in hoofdstuk 9. De functionaliteit van EPAK lijkt erg op die van SIX. De verschillen zitten in het feit dat men bij de ontwikkeling van EPAK wel de papierwensen als uitgangspunt genomen heeft. Verder is het zo dat de aard van de beschikbare kennis verschilt. Bij Vertis/AVEBE is de kennis voornamelijk beschikbaar in de vorm van vuistregels terwijl bij EPAK vooral voor het samenstellen van de kwaliteitsmatrix de kennis erg concreet aanwezig is. Van een beschikbaar mathematisch model is ook bij SIX (nog) geen sprake, daarnaast lijkt het op basis de de eerste interviews niet mogelijk binnen de looptijd van het project voldoende kennis te vergaren voor het samenstellen van een dergelijke kwaliteitsmatrix.

6. PROBLEEMANALYSE

R.A. Carton

Voor de verwezenlijking van het expertsysteem zijn verschillende oplossingen mogelijk. Natuurlijk heeft iedere oplossing voor– en nadelen. In paragraaf 6.1. is een aantal mogelijkheden genoemd. Omdat de gewenste implementatievorm al in de opdrachtomschrijving is aangegeven, hoeft er geen keuze meer te worden gemaakt. Voor de volledigheid zijn hier toch enkele alternatieven genoemd. Bovendien wordt iets gezegd over de gebruikte oplossingsmethode.

6.1. Oplossingsmethoden

Hieronder worden vier principiële mogelijkheden beschreven waarmee het expertsysteem kan worden geïmplementeerd. De eerste vier oplossingen die behandeld worden, zijn achtereenvolgens: een database, een *neuraal netwerk*, *fuzzy logic* en een expertsysteem. Van deze mogelijke implementatievormen worden de belangrijkste voor– en nadelen opgesomd, voorzien van commentaar. Daarna volgt een beschrijving van een *hybride* oplossing. Het versmelten van methoden uit verschillende vakgebieden komt in de praktijk steeds vaker voor.

6.1.1. Database

Het gebruik van een database om het probleem op te lossen heeft een aantal voordelen, maar in het algemeen schiet een dergelijke implementatie te kort op het gebied van redeneren. De voordelen van een database-oplossing voor het onderhavige geval zijn:

+ de gegevens van het Papierlab bevinden zich in een database
+ bij Vertis is het database-pakket ORACLE beschikbaar voor implementatie
+ de implementatie van een database-applicatie is relatief eenvoudig

De resultaten van de tests uit het Papierlab van AVEBE worden opgeslagen in een ORACLE-database. Aanvankelijk werd dit als voordeel gezien, omdat deze gegevens direct door het te ontwerpen programma gebruikt zouden kunnen worden. Later bleek echter, dat de rapporten met resultaten niet het geschikte formaat hadden voor gebruik door een programma. Dit punt is dus komen te vervallen. Een echt voordeel is de ervaring die Vertis met ORACLE heeft. Een ander pluspunt is dat het toepassen van regels op de gegevens uit een database vrij eenvoudig is.

De keuze voor een puur database-gerichte oplossing brengt ook nadelen met zich mee, waarvan de belangrijkste hieronder staan:

- de expertkennis moet worden geformaliseerd
- de kennisstructuur is niet inzichtelijk
- het systeem kan keuzes niet beargumenteren
- het systeem kan geen nieuwe oplossingen geven
- databases werken niet met onzekerheid

De kennis van de experts van AVEBE moet worden vastgelegd en ingebracht in het systeem. Omdat het systeem daarvoor IF-THEN regels zal gebruiken, zal de structuur van deze kennis niet duidelijk terug te vinden

zijn. Een ander nadeel van het werken met een conventionele database is het gebrek aan automatische onderbouwing van een bepaalde berekening door het systeem. Het programma kan ook slechts een van te voren vastgelegde uitvoer genereren. Het zoeken naar nieuwe, betere oplossingen is dus niet mogelijk. Verder is het werken met (on)zekerheden niet inherent aan een database-systeem; de implementatiefase wordt dus langer als er betrouwbaarheidsfactoren moeten worden toegekend aan de gegeven oplossingen.

6.1.2. Neuraal netwerk

Een heel andere aanpak van het probleem is het gebruik van een neuraal netwerk . Deze vorm van *kunstmatige intelligentie* heeft enkele grote voordelen, te weten:

+ het probleem kan worden herleid tot dat van patroonherkenning
+ de expertkennis hoeft niet te worden geformaliseerd
+ het systeem kan nieuwe oplossingen geven
+ de vastgelegde gegevens worden alleen tijdens het leerproces gebruikt

Het zoeken naar verbeteringen voor het proces kan worden herleid tot een patroonherkennings probleem en een neuraal netwerk is uitermate geschikt voor het oplossen van dergelijke problemen. Bovendien hoeft er geen structurering van de kennis van de experts plaats te vinden, omdat het netwerk de gegevens zelf structureert. Verder kan het systeem oplossingen genereren die het niet expliciet heeft geleerd. Het laatstgenoemde voordeel betreft geheimhouding: de gegevens van AVEBE over de papierfabricageprocessen van klanten worden alleen in de leerfase door het programma gebruikt. Ze hoeven dus niet bij het systeem gevoegd te worden.

De nadelen van een neuraal netwerk als afleidingssysteem zijn de volgende:

- de kennisstructuur is onduidelijk
- het systeem kan keuzes niet beargumenteren
- de vastgelegde gegevens zijn misschien niet toereikend

De structuur van de kennis die het neurale netwerk heeft is onduidelijker dan bij een database. Gegenereerde oplossingen kunnen, net als bij een database-implementatie, niet worden onderbouwd. Verder is voor het leerproces van een netwerk een verzameling gegevens nodig, die een zo volledig mogelijk beeld van het probleem en de oplossingen geeft. De gegevens die bij het Papierlab van AVEBE aanwezig zijn, zouden in diversiteit wel eens ontoereikend kunnen zijn om het netwerk goed te trainen.

6.1.3. Fuzzy logic

Met behulp van fuzzy logic zou ook een implementatie kunnen worden gedaan. Dit brengt tenminste drie voordelen met zich mee, namelijk:

+ onzekerheid/vaagheid is inherent aan fuzzy logic
+ het systeem kan nieuwe oplossingen geven
+ de kennisstructuur is vrij inzichtelijk

Omdat fuzzy logic draait om vaagheid, kan het systeem betrouwbaarheidswaarden geven voor de gegenereerde oplossingen. Het systeem kan ook oplossingen geven die niet expliciet in de regels voorkomen. Verder geeft de vorm van de regels een redelijk duidelijk inzicht in de kennis van het systeem, hoewel het overzicht bij een groot aantal regels makkelijk verloren gaat.

Een fuzzy systeem heeft natuurlijk ook een schaduwzijde. De minpunten zijn:

- de kennis moet worden geformaliseerd
- het systeem kan keuzes niet beargumenteren

Net als bij een database-oplossing moet er structuur in de kennis van de experts worden aangebracht en kan het systeem niet aangeven waarom er voor een bepaalde oplossing is gekozen.

6.1.4. Expertsysteem

Een expertsysteem als implementatievorm voor het programma brengt de volgende voordelen met zich mee:

+ het systeem kan gemaakte keuzes onderbouwen
+ de kennisstructuur is duidelijk
+ het systeem kan de betrouwbaarheid van oplossingen geven
+ de regelverzameling is vrij makkelijk uitbreidbaar
+ G2 is beschikbaar voor implementatie

Bij iedere gegenereerde oplossing kan het systeem aangeven welke regels daartoe aanleiding hebben gegeven. Bovendien blijkt uit de regels welke kennis er is vastgelegd. Oplossingen worden voorzien van een betrouwbaarheidsfactor die aangeeft hoe goed ze zijn. Omdat de regels makkelijk leesbaar zijn, is het relatief eenvoudig de verzameling uit te breiden. Binnen Vertis is voor implementatie de expertsysteem-shell G2 beschikbaar.

Het nadeel van het gebruik van een expertsysteem is:

- de kennis moet worden geformaliseerd

In tegenstelling tot wat bij neurale netwerken het geval is, moet er structuur worden aangebracht in de kennis van de experts.

6.1.5. Hybride systeem

De hiervoor behandelde oplossingen zullen in de praktijk vaak gecombineerd worden. Een expertsysteem kan bijvoorbeeld worden voorzien van een neurale uitbreiding, waarmee het in staat is te leren van situaties uit het verleden. Verder worden de gegevens waarover een expertsysteem moet redeneren meestal opgeslagen in een database (en dit geldt soms ook voor de regels in het systeem). Bij SIX wordt min of meer van fuzzy logic gebruik gemaakt (zie de paragraaf "Initialisatie– en classificatiekennis" in § 10.2.2.): het indelen in klassen van de gegevens kan worden beschouwd als een "fuzzificatie"-proces. Fuzzificatie is het omzetten van numerieke waarden naar fuzzy waarden; bijvoorbeeld: als de gemeten waarde voor de pH 3 is, zou daarbij de fuzzy waarde "erg zuur" kunnen horen.

6.2. Gebruikte methode

De keuze van de implementatievorm voor SIX lag voor de hand, aangezien in de doelstelling staat dat G2 zal worden gebruikt. SIX is een expertsysteem omdat G2 is gemaakt voor de ontwikkeling van dit soort systemen. Door de vorm van de kennis (IF-THEN-regels) is de structuur vrij inzichtelijk. De verzameling regels is (voor iemand met kennis van G2 en SIX) redelijk makkelijk uitbreidbaar. Als in een G2-applicatie met betrouwbaarheidswaarden wordt gewerkt (iets dat het systeem ondersteunt), kan deze aangeven hoe "waar" een bepaalde uitspraak is en dus hoe betrouwbaar. Bij SIX wordt hiervan overigens geen gebruik gemaakt. Ondanks het feit dat G2 tijdens het redeneren kan laten zien welke regels er "vuren" (worden geactiveerd), is het standaard niet mogelijk om bij een conclusie een lijstje daarvoor verantwoordelijke regels te geven. Dit zou expliciet door de programmeur aangebracht moeten worden. Het beargumenteren van oplossingen is dus geen onderdeel van deze ontwikkelomgeving. Hoofdstuk 9 behandelt G2 uitgebreid.

7. KENNISREPRESENTATIE EN EXPERTSYSTEMEN

M. Pruis

In dit hoofdstuk komen, als aanvulling op de informatie uit het vorige hoofdstuk, op een beknopte wijze twee bestaande methoden om kennis te representeren in een expertsysteem aan bod. Eerst wordt de kennisrepresentatie met behulp van logica besproken, eigenlijk de eerste serieuze methode die gebruikt werd door de kunstmatige intelligentie. Daarna komt de theorie van het produktiesysteem en pr oduktieregels aan bod. In dit hoofdstuk wordt de meeste aandacht besteed aan deze laatste theorie omdat deze door de vaststaande keuze van de ontwikkelomgeving opgelegd was. De kennis die in deze paragraaf aan bod komt is niet direct noodzakelijk voor het gebruik van G2 omdat een aantal aspecten door G2 zelf onder water geregeld wordt. Deze paragraaf vormt een basis voor het algemeen begrip van de werking van G2 en problemen die zich kunnen voordoen bij expertsystemen in het algemeen. Tot slot zal bekeken worden hoe onzekerheid verwerkt kan worden in de representatie met behulp van produktieregels en in hoeverre dit van belang is voor SIX. De voorbeelden in dit hoofdstuk zijn zoveel mogelijk afgeleid uit het papierfabricage-domein zodat er meteen een idee ontstaat van de bruikbaarheid van de verschillende methodieken. Voorbeelden die niet uit dit domein zijn gekozen (omdat er geen duidelijke voorbeelden voorhanden waren) zijn overgenomen uit [Rich, 91] of [Lucas, 88].

7.1. Logica

Het *formalisme* van de logica is aantrekkelijk voor het representeren van kennis omdat daar direct aan gekoppeld een krachtige manier bestaat om nieuwe kennis uit oude af te leiden: mathematische deductie [Rich, 91]. Vrijwel direct nadat de eerste digitale computers beschikbaar kwamen (begin jaren '50) werd een begin gemaakt met onderzoek dat het toepassen van logica voor het met de computer oplossen van problemen tot doel had. Een klassiek voorbeeld op het gebied van theorem proving, de Logic Theorist, werd in 1955 ontwikkeld door A. Newell, J.C. Shaw en H.A. Simon. Dit programma was in staat een aantal stellingen uit de "Principia Mathematica" van A.N. Whitehead en B. Russel te bewijzen. Al vroeg werd echter duidelijk dat de standaard afleidingsregels van de logica minder geschikt waren voor de computer [Lucas, 88]. Door de ontwikkeling van resolutie, een voor de computer beter geschikte en efficiëntere afleidingsregel, heeft theorem proving toch aan belang binnen de informatica gewonnen. In deze paragraaf zal eerst kort aandacht worden besteed aan propositie– en predikatenlogica. In 7.1.2. wordt globaal de werking van de resolutiemethode beschreven als methode van redeneren.

7.1.1. Propositie– en predikatenlogica

Propositielogica houdt zich bezig met uitspraken die óf waar óf onwaar zijn, bijvoorbeeld: "Amylofax is een zetmeel" of "Amylofax is kationisch". Dit soort uitspraken worden meestal met een hoofdletter genoteerd en proposities genoemd. De enkelvoudige proposities P en Q worden ook wel atomaire proposities of atomen genoemd. Met behulp van logische operatoren of connectieven kunnen samengestelde proposities opgesteld worden. In de logica worden vijf operatoren gebruikt:

- negatie ($\neg$)
- conjunctie ($\wedge$)

- disjunctie ($\vee$)
- implicatie ($\rightarrow$)
- bi-implicatie ($\leftrightarrow$)

Als we bijvoorbeeld uitgaan van de proposities P en Q, die de volgende betekenis hebben:

P = "Amylofax is een zetmeel"
Q = "Amylofax is kationisch"

dan heeft de samengestelde propositie $P \wedge Q$ de betekenis:

"Amylofax is een zetmeel en amylofax is kationisch"

Formules die alleen bestaan uit atomen en connectieven (samengestelde proposities) en syntactisch correct zijn, worden welgevormde formules genoemd. Ter volledigheid moet gezegd worden dat niet in alle literatuur het gebruik van hoofdletters voor atomen gebruikelijk is. In [Rich, 91] bijvoorbeeld worden atomen ook voorgesteld door een reeks hoofdletters waaraan een betekenis gekoppeld kan worden. In bovenstaand voorbeeld zouden dan de atomen AMYLOFAXZETMEEL en AMYLOFAXKATIONISCH zijn ontstaan. De betekenis is voor de logica niet van belang maar kan handig zijn voor diegene die ermee werkt.

Al snel blijkt een belangrijk nadeel van propositielogica. Wanneer we ook willen representeren dat Solvitose een zetmeel is, zouden we dat representeren door SOLVITOSEZETMEEL. Dit zou een compleet afzonderlijke propositie zijn en uit de kennis die we gerepresenteerd hebben kunnen we niks concluderen over overeenkomsten tussen Amylofax en Solvitose. Dit zou beter weergegeven kunnen worden door iets op te schrijven als:

zetmeel(amylofax)
zetmeel(solvitose)

Het wordt nog moeilijker als we vervolgens de propositie ZETMEELRETENTIEMIDDEL opstellen als representatie van "alle zetmelen zijn retentiemiddel". Ondanks dat we aangegeven hebben dat alle zetmelen gebruikt kunnen worden als retentiemiddel en we weten dat Amylofax een zetmeel is kunnen we niet het verband leggen dat Amylofax als retentiemiddel gebruikt kan worden. Om dit wel mogelijk te maken moeten we gebruik kunnen maken van variabelen en kwantoren zoals die aanwezig zijn in de eerste orde predikatenlogica. De predikatenlogica maakt gebruik van de volgende symbolen:

- Predikaten - definieert een relatie tussen *n* objecten, n-plaatsig
- Functie - geeft een waarde uit het bereik voor *n* argumenten, n-plaatsig
- Variabelen - beginnend met een hoofdletter
- Constanten - beginnend met een kleine letter
- Logische connectieven - $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$
- Kwantoren - $\forall$ (voor alle), $\exists$ (er is een)

Wanneer een predikaat nul-plaatsig is noemen we dit een atomaire propositie, één, twee– en drieplaatsige predikaatsymbolen noemt men respectievelijk ook wel unaire, binaire en ternaire predikaatsymbolen. Voorbeelden van predikaatsymbolen zijn:

onderdeel_van(headbox, papiermachine)
reageert_met(amylofax, alum)
is_gelijk_aan(7, 7)

Deze geven dus duidelijk een relatie aan tussen objecten. Een functie "mapt" argumenten (objecten) op een element uit een bepaald bereik en heeft als waarde dat element. Voorbeelden van mogelijke functies zijn:

plus(2, 3)
pH(amylofax)

En met behulp van de kwantoren is het mogelijk om de zin "alle zetmelen zijn retentiemiddel" weer te geven:

$\forall x$: zetmeel(x) $\rightarrow$ retentiemiddel(x)

Voor het redeneren met behulp van logica zijn er zogenaamde afleidings- of inferentieregels gedefinieerd. Dit zijn zuiver syntactische operaties die slechts de vorm van formules wijzigen. Een afleidingsregel wordt in het algemeen als een schema weergegeven, zoals bijvoorbeeld:

$$\frac{A, A \rightarrow B}{B}$$

De formules boven de streep worden premissen genoemd, en het deel onder de streep is de conclusie. Met behulp van deze afleidingsregel (die de *modus ponens* genoemd wordt) kan een implicatie worden verwijderd. Een voorbeeld van een regel waarmee een logisch connectief kan worden geïntroduceerd is de volgende:

$$\frac{A, B}{A \wedge B}$$

Door middel van het herhaald toepassen van afleidingsregels onstaat een afleiding. Met behulp van de modus ponens bijvoorbeeld kan uit de formules $P \wedge (Q \vee R)$ en $P \wedge (Q \vee R) \rightarrow S$ de conclusie S getrokken worden. Men zegt dan dat er een afleiding of deductie van S uit de verzameling formules bestaat. Overigens kunnen deze afleidingsregels zowel bij propositie- als predikatenlogica gebruikt worden.

Het is duidelijk dat manier van representeren met behulp van predikatenlogica een stuk krachtiger is dan met behulp van propositielogica. Jammer genoeg bestaat er voor predikatenlogica geen beslisprocedure, in tegenstelling tot bij propositielogica. Er bestaan wel procedures die een bepaalde theorie kunnen bewijzen, mits de theorie bewijsbaar is. Als de theorie niet bewijsbaar is, is er geen garantie dat deze procedures tot een conclusie komen (waar of niet waar). Predikatenlogica is dus niet *beslisbaar* maar *semi-beslisbaar*. Een simpel voorbeeld is een procedure die domweg in een bepaalde volgorde inferentiemechanismen toepast op de axioma's totdat het resultaat gelijk is aan de te bewijzen theorie. Deze methode is niet bijster efficiënt en binnen de kunstmatige intelligentie zijn al betere, intelligentere methoden ontwikkeld. Eén van de uitgangspunten is daarbij geweest dat het weliswaar zo is dat we niet een altijd beslissende procedure kunnen ontwikkelen, maar dat dat niet wil zeggen dat er geen methode ontwikkeld kan worden die voor bijna alle invoer een beslissing heeft.

Ondanks het theoretisch niet beslisbaar zijn van de predikatenlogica lijkt het wel heel bruikbaar te zijn bij het representeren en manipuleren van kennis. Daarbij moet een aantal punten opgemerkt worden waarmee rekening gehouden moet worden bij het gebruik van logica:

- Er zijn vaak verschillende manieren om kennis te representeren met behulp van logica. Zo eenvoudig mogelijk geniet natuurlijk de voorkeur maar bepaalde manieren van representeren kunnen bepaalde manieren van redeneren uitsluiten. Hierbij geldt dat een zo algemeen mogelijke structuur van de kennis meestal de meeste mogelijkheden biedt maar ook het meest complex wordt.
- De complexiteit van de kennisstructuur neemt vaak enorm toe omdat er erg veel bijna triviale kennis en niet direct aan het probleem gerelateerde kennis over de wereld noodzakelijk is om bepaalde afleidingen te kunnen doen.
- Zelfs voor het afleiden van heel simpele conclusies kan het zo zijn dat er enorm veel redeneerstappen voor het bewijs gemaakt moeten worden (zie ook vorige punt).
- Voor het vormen van een bewijs wordt een aantal complexe methoden zoals *matching*, *substitutie* en toepassing van het genoemde modus ponens gebruikt. Verder beoordeelt de mens tijdens het afleiden van een bewijs zelf (op basis van ervaring, oefening) welke stappen het best genomen kunnen worden.

Het eerste punt is een belangrijk aandachtspunt tijdens de ontwikkeling van een systeem. Afhankelijk van de benodigde functionaliteit moet hier een afweging gemaakt worden tussen inzichtelijkheid, complexiteit en mogelijkheden en efficiëntie van gekozen kennisstructuur. Het tweede en derde punt impliceren dat wanneer een systeem op een efficiënte manier meer moet kunnen dan alleen triviale dingen afleiden het op een bepaald moment grotere stappen moet gaan nemen bij het bewijzen. Mensen worden ook langzamerhand experts en leren ook langzamerhand om de basis afleidingen over te slaan en die gewoon voor waar aan te nemen. Uit het laatste punt kan opgemaakt worden dat het bouwen van een programma dat op dezelfde wijze als de mens afleidingen doet heel complex is. In de volgende paragraaf wordt de basis van resolutie beschreven. Deze methode beperkt de complexiteit van het redeneerprobleem omdat het alleen afleidingen maakt met behulp van

statements die opgesteld zijn in een bepaalde standaardvorm en er geen keuze gemaakt hoeft te worden welke inferentiestap toegepast moet worden.

7.1.2. Resolutie

Resolutie is gebaseerd op het feit dat alle statements waarmee bewijzen afgeleid kunnen worden, op een bepaalde handige standaardvorm zijn opgesteld. Deze standaardvorm wordt clausal form genoemd, waarbij alle formules omgezet worden naar een verzameling clauses die alle disjuncties van *literals* zijn (zie [Lucas, 88], [Rich, 91] voor een omzettingsprocedure naar clausal form).

Resolutie is een simpel *iteratief* proces. Bij elke stap worden twee clauses, de zogenaamde parent-clauses, vergeleken (resolve) er een nieuwe clause (resolvent) toegevoegd die weergeeft hoe de parent-clauses elkaar beïnvloeden. Stel dat er twee clauses aanwezig zijn in het systeem:

Winter ∨ Zomer
¬Winter ∨ Koud

Nu kunnen we constateren dat of Winter of ¬Winter waar zal zijn op een bepaald moment. Als Winter waar is dan zal Koud ook waar moeten zijn om de waarheid van de tweede clause te garanderen. Als ¬Winter waar is dan zal Zomer waar moeten zijn om de waarheid van de eerste clause te garanderen. Van deze twee clauses kan dus afgeleid worden dat:

Zomer ∨ Koud

Dit is het soort deducties dat de resolutiemethode maakt. Resolutie neemt twee clauses waarin dezelfde literal voorkomt (eenmaal positief en een keer negatief). De resolvent is afgeleid door alle literals uit de parent-clauses te combineren behalve die tegen elkaar wegvallen. Als er een lege clause afgeleid wordt (bijvoorbeeld uit de twee clauses Winter en ¬Winter), dan is er een contradictie gevonden. Wanneer de bovenstaande deducties systematisch uitgevoerd worden en er een contradictie aanwezig is, dan is gegarandeerd dat die gevonden wordt. Als de contradictie niet bestaat, is echter niet gegarandeerd dan de procedure eindigt (*onbeslisbaar*). Resolutie werkt door af te leiden dat de tegenstelling van de te bewijzen formule een contradictie genereert. Bovenstaande geeft slechts een indruk van de basis van de resolutiemethode. De lezer wordt verwezen naar [Lucas, 88], [Rich, 91] voor een uitgebreidere verhandeling.

Sommige kennis kan erg moeilijk weergegeven worden met behulp van logica ("het is *erg* warm vandaag", "de waarde van x is *vaak* te hoog", etc.) verder is logica een vrije starre methode. Er moet zoals gezegd erg veel algemene kennis opgeslagen worden, zelfs voor de simpelste afleidingen. Hierdoor is er binnen de *AI* al veel onderzoek gedaan naar andere methoden voor representatie. Logica blijkt echter ook tegenwoordig nog een interessant en regelmatig toegepast formalisme.

7.2. Produktieregels en inferentie

In het begin van de jaren 70 is door A. Newell het begrip produktiesysteem geïntroduceerd als model van het menselijk redeneervermogen [Newell, 73]. In dit model wordt ervan uitgegaan dat de menselijke kennis uit min of meer onafhankelijke eenheden bestaat, produktieregels genaamd. Deze produktieregels die ook wel produkties genoemd worden bevatten informatie over de acties die uitgevoerd moeten worden als er een bepaald patroon herkend wordt in de omgeving. Deze acties hebben weer effect op de waarneming van de omgeving omdat bepaalde aannames verworpen kunnen worden of nieuwe veronderstellingen gemaakt kunnen worden.

Veel van de huidige expertsystemen maken in verschillende vormen gebruik van dit formalisme van de produktieregel voor kennisrepresentatie. De ervaring met dit soort systemen heeft aangetoond dat produktieregels met name bij het oplossen van classificatieproblemen, waarvoor de benodigde kennis de vorm heeft van vuistregels, een goed hulpmiddel bieden [Lucas, 88].

De produktieregel is het belangrijkste onderdeel van een produktiesysteem. Met behulp van een produktieregel kan *heuristische* probleemkennis worden vastgelegd. Een verzameling van deze regels wordt de regelverzameling (rule base) van het produktiesysteem genoemd. Verder kent een produktiesysteem een formalisme voor de gegevensverzameling zodat de objecten waarover geredeneerd wordt en hun relaties beschreven kunnen worden. Dit formalisme wordt gegevensbeschrijving genoemd en vormt samen met de rule base de knowledge base.

7.2.1. Globale gegevensverzameling

Naast de knowledge base bevat een produktiesysteem ook nog een feitenverzameling of globale gegevensverzameling [Lucas, 88]. Deze bevat de feitelijke informatie in een produktiesysteem waarmee geredeneerd wordt. Gedurende de consultatie van het systeem wordt nieuwe informatie afgeleid of door de gebruiker ingevoerd. Deze informatie is de invulling van de gegevensbeschrijving uit de knowledge base. Volgens [Lucas, 88] is deze gegevensverzameling geen onderdeel van de knowledge base maar een apart deel van het systeem omdat de informatie afhankelijk is van de sessie op dat moment. Door sommige andere auteurs wordt de gegevensverzameling (of een deel daarvan) wel tot de knowledge base gerekend. Hierbij wordt dan de nadruk gelegd op de initiële gegevensverzameling die de kennis bevat die niet in regels vastgelegd wordt. Bepaalde vaststaande feiten of default informatie is op die manier opgenomen in de gegevensverzameling en behoort ook tot de knowledge base.

De informatie in de gegevensverzameling kan op verschillende manieren gerepresenteerd worden, afhankelijk van de gegevensbeschrijving. Eén manier is het gebruik van *variabelen* die tijdens een sessie één of meer waarden kunnen aannemen. Over het algemeen worden er twee soorten variabelen gebruikt namelijk de *meerwaardige* (multivalued) en de *eenwaardige* (singlevalued) *variabelen*. Een variabele met de waarden die tijdens de sessie (of consultatie in [Lucas, 88]) bekend zijn of toegekend zijn wordt een feit genoemd. Voorbeelden van feiten (met eenwaardige en meerwaardige variabelen) zijn:

```
klant            =  AVEBE
gebruikt_zetmeel =  Amylofax 2200
gebruikt-hulpstof =  {Alum, Kleurstof}
```

7.2.2. Knowledge base

In de regelverzameling is kennis gerepresenteerd door middel van produktieregels waarin een aantal uitspraken met elkaar in verband worden gebracht. Dit kan gebeuren door middel van als...dan... regels. In de meest eenvoudige vorm zijn de condities en conclusies in zo'n regel uitspraken over variabelen en hun waarde (mogelijke feiten). Door middel van predikaten en acties in de condities en/of conclusies van de regels kunnen complexere verbanden tussen uitspraken gelegd worden. In [Lucas, 88] wordt als voorbeeld "same" gebruikt als voorbeeld van een predikaat. Deze test of een bepaalde (meerwaardige) variabele (o.a.) een bepaalde waarde heeft aangenomen. Als voorbeeld van een actie wordt "assign" genoemd. Deze kent een bepaalde meegegeven waarde toe aan een meegegeven variabele. Een voorbeeld van een mogelijke regel voor SIX is de onderstaande produktieregel:

```
if
  greaterthan(cationic-demand, 80) and
  smallerthan(starch-retention, 70%)
then
  assign(problem, anionic-trash)
```

7.2.3. Inferentie in een produktiesysteem

Het is gebruikelijk om onderscheid te maken tussen twee basisvormen van automatisch redeneren. Men spreekt van top-down inferentie of doel-gestuurde inferentie wanneer gebruik gemaakt wordt van het doel van de redenering om subdoelen af te leiden om op deze manier op de initiële gegevens uit te komen. Deze methode wordt gebruikt in een *backward reasoning* systeem. De tweede basisvorm is bottom-up inferentie of data-gestuurde inferentie. Hierbij wordt uitgegaan van de initiële gegevens en wordt getracht zoveel mogelijk nieuwe

informatie af te leiden [Lucas, 88]. Deze methode is de basis van *forward reasoning* systemen [Rich, 91]. Bij de eerste methode zoekt het systeem naar passende subdoelen in de conclusies van de produktieregels, in het tweede geval worden de initiële gegevens gematcht op de condities in de produktieregels.

Bottom-up inferentie versus top-down inferentie

Omdat de feiten en produktieregels in beide methoden verschillend gebruikt worden zijn de twee methoden ook geschikt voor verschillende soorten expertsystemen. Top-down inferentie wordt vaak gebruikt bij diagnostische systemen waarbij de inferentie gestuurd kan worden door een doel en een geringe hoeveelheid invoergegevens. Bottom-up inferentie is het meest geschikt voor toepassingen waarbij de interpretatie van grote hoeveelheden gegevens een rol speelt en a priori niet duidelijk is welke doelen bereikt dienen te worden. In het algemeen kan gezegd worden dat er vier factoren zijn die bepalen welke inferentie (redeneer) methode efficiënter is [Rich, 91]:

- Zijn er meer mogelijke begintoestanden of meer eind(doel)toestanden? (Waarbij een toestand, volgens de in dit hoofdstuk gebruikte naamgeving uit [Lucas, 88], gedefinieerd wordt als een verzameling feiten op een bepaald moment) In een systeem is het makkelijker om van een klein aantal toestanden naar een groter (en makkelijker te vinden) aantal toestanden te redeneren.

- In welke richting is het aantal mogelijke afsplitsingen in de redenering groter? Het is efficiënter om te redeneren in de richting met het minst aantal mogelijke afsplitsingen.

- Moet het systeem verklaren (aan de gebruiker) hoe het tot de conclusie gekomen is? Als dit het geval is is het belangrijk om de richting aan te houden die het meest overeenkomt met de denkwijze van de gebruiker.

- Welke gebeurtenissen (*events*) zorgen ervoor dat een redenering (of een deel van de redenering) gestart wordt? Als dit het bekend worden van een nieuw feit is, dan is het logisch om forward reasoning te gebruiken (bottom-up inferentie). Wanneer de redenering opgestart wordt door een vraag van de gebruiker dan is het natuurlijker om een backward reasoning systeem te maken.

Wanneer we uitgaan van de representatie met behulp van variabelen en waarden dan is het doel in een top-down inferentie het toekennen van de waarde(n) aan de zogenaamde doelvariabele. Het redeneren gebeurt dus vanuit de te trekken conclusie. Het bepalen van waarden van een variabele wordt *traceren* genoemd. Dit betekent dat er voor het bepalen van de waarde van een variabele net zo lang regels toegepast worden totdat alle produktieregels die kunnen leiden tot een waarde voor die variabele zijn gebruikt. In het systeem kan onderscheid gemaakt worden tussen variabelen die vraagbaar (askable) zijn en variabelen die niet gevraagd mogen worden. Wanneer het systeem met behulp van de regels niet in staat is om de waarde van een variabele af te leiden wordt deze in het geval van een vraagbare variabele aan de gebruiker gevraagd.

In tegenstelling tot de top-down inferentie wordt bij bottom-up inferentie uitgegaan van de feitenverzameling waarna aan de hand van de beschikbare feiten regels worden toegepast die weer leiden tot nieuwe feiten in de feitenverzameling. Er wordt dus uitgegaan van de feiten en geredeneerd naar eventuele conclusies. Ook bij bottom-up inferentie worden een aantal relevante produktieregels geselecteerd die samen de *conflict set* vormen. Relevante produktieregels zijn in dit geval regels waarvan aan alle condities is voldaan op basis van de feitenverzameling. Van deze relevante regels wordt er één uitgekozen en toegepast. Omdat een bepaalde actie ook een "unassign" kan zijn is het mogelijk dat er na het toepassen van de conclusie niet meer aan alle condities van de andere regels uit de conflict set voldaan wordt. Daarom wordt er dus steeds maar één regel uitgevoerd. In de volgende inferentiestap worden de overige regels uiteraard opnieuw bekeken. Dit proces wordt herhaald totdat een bepaald stopcriterium behaald is. Meestal is dat het leeg zijn van de conflict set zodat er niks meer afgeleid kan worden.

In principe kunnen dezelfde regels gebruikt worden voor forward (bottom-up) reasoning en backward (top-down) reasoning. In praktijk is gebleken dat het handig is om de regels onder te verdelen in twee klassen, die elk een bepaald soort kennis kunnen bevatten [Rich, 91]:

- Forward-chaining regels - deze regels bevatten kennis over de juiste acties als gevolg van bepaalde feiten.
- Backward-chaining regels - deze regels bevatten kennis over de juiste acties om een bepaald doel te bereiken.

Een simpele methode bij backward-chaining regels om te bepalen welke produktieregels kunnen worden toegepast (matchen van regels) is kijken of de doelvariabele voorkomt in één van de conclusies van een andere produktieregel. De aldus geselecteerde regels kunnen dan worden toegepast. De verzameling van geselecteerde produktieregels is de conflict set. Wanneer de volgorde van het toepassen van deze regels niet vast ligt noemt [Lucas, 88] dit non-determinisme van de eerste soort, als de volgorde van het evalueren van de condities uit de geselecteerde regels niet vast ligt non-determinisme van de tweede soort. Als de toepassing van de regels plaatsvindt in de volgorde van specificatie in de regelverzameling, en dit ook geldt voor de evaluatie van de condities in een regel, dan is dit de meest simpele vorm van backward-chaining. Hetzelfde geldt voor een simpele vorm van forward-chaining. Twee belangrijke factoren in de efficiëntie en het karakter van een systeem zijn de methoden waarmee het systeem de matching verricht en *conflictresolutie* toepast.

7.2.4. Matching

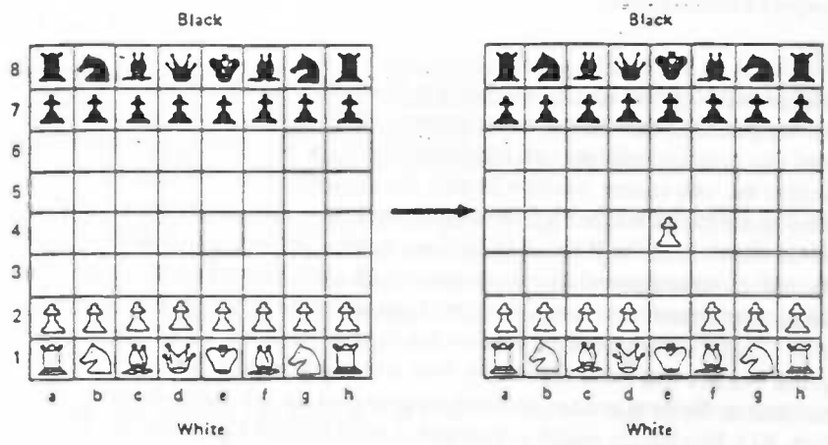
Tot nu toe is het redeneren uitgelegd als zijnde het selecteren van de juiste regels bij een bepaalde toestand, die toe te passen zodat er een nieuwe toestand ontstaat. Dit totdat er een bepaalde eindtoestand ontstaat. Naast het selecteren van de juiste regel uit de conflictset kan ook het samenstellen van deze conflictset (het matchen van regels en toestand) een belangrijke factor zijn. De hierboven genoemde simpele methode waarbij alle mogelijk toepasbare regels opgezocht worden door de condities te vergelijken met de huidige toestand kent twee belangrijke problemen [Rich, 91]:

- Wanneer de systemen in staat moeten zijn complexe problemen op te lossen zal het aantal regels al gauw erg groot worden. Elke keer alle regels doorlopen om te bepalen welke er toegepast kunnen worden is erg inefficiënt.

- Het is niet altijd direct duidelijk of aan de condities in een regel voldaan is in een bepaalde toestand.

Matching met indexering

In sommige gevallen kan het eerste probleem gemakkelijk opgelost worden door een index te genereren van condities en de regels die die condities bevatten. Wanneer er echter gebruik wordt gemaakt van algemene regels in een systeem, en dit is bij complexere systemen wel gewenst, wordt het indexeren al veel moeilijker. Aan de hand van een voorbeeld van een schaakspel blijkt dit al snel [Rich, 91]. De regel zoals afgebeeld in figuur 7.1. definieert een mogelijk zet aan het begin van het spel. In principe kunnen alle zetten die dan mogelijk zijn opgeslagen worden met als index alle posities van de stukken op het bord[4]. Deze methode werkt alleen omdat de condities in de regel exact overeenkomen met bordsituaties. Het grote nadeel is dat er per regel veel specifieke maar overbodige informatie opgeslagen wordt. Het voordeel is dat de situatie direct op de indexering afgebeeld kan worden.



Figuur 7.1. Mogelijke representatie van een schaakzet

[4] De bord-opstelling zou gerepresenteerd kunnen worden als een grote integer. Door middel van een redelijke hashing-functie kan dat nummer als index dienen zodat alle bord-posities snel gevonden kunnen worden.

Wanneer we de regel algemener zouden willen houden (zie fig. 7.2.) dan kan er niet meer geïndexeerd worden op exacte bordsituatie. Het is wel gewenst om de regels op de tweede manier op te slaan omdat we in het eerste geval één zet bij al zijn mogelijke bordsituaties moeten opslaan.

IF
WittePion op positie(e2)
AND
positie(e3) is leeg
AND
positie(e4) is leeg

→ Verplaats Pion van positie(e2) naar positie(e4)

Figuur 7.2. Een algemene manier om een schaakzet te beschrijven

Bovenstaand voorbeeld kan ook vertaald worden naar het SIX probleem. Wanneer we alle mogelijke combinaties van meetwaarden zouden willen opnemen met daarbij de adviezen die gegeven moeten worden of aanpassingen in het proces die gemaakt zouden moeten worden dan zouden er erg veel regels opgeslagen moeten worden. Figuur 7.3 is een voorbeeld van een mogelijke algemene regel voor het papierfabricageproces. Om deze regel te kunnen toepassen zijn er slechts drie eigenschappen van belang. Wanneer we de regels op deze wijze opslaan is het niet meer mogelijk om op de standaard wijze te indexeren maar deze enkele regel kan wel op veel situaties toegepast worden.

IF
Cationic-demand in headbox > 80
AND
Zetmeelretentie < 70 %
AND
Trash-catcher ∉ Toevoegingen

→ Voeg toe trash-catcher voor headbox

Figuur 7.3. Een mogelijke regel voor het papierfabricageproces

Wanneer er algemene (high-level) regels gebruikt worden betekent dat niet meteen dat indexering van de regels helemaal niet meer nuttig zou zijn. In veel systemen (bijvoorbeeld ook PROLOG) worden de regels geïndexeerd op de predikaten die gebruikt worden in de regels. Op deze manier wordt het al weer makkelijker om de regels te selecteren die toepasbaar zouden kunnen zijn. In bovenstaande voorbeeld zouden de regels geïndexeerd kunnen worden op de pulp & zetmeeleigenschappen die van toepassing zijn en de actie die ondernomen wordt. Ondanks de beperkingen bij deze methode is het indexeren van regels (in welke vorm dan ook) belangrijk voor de effieciëntie van het expert/kennissysteem.

Matching met variabelen

Het selecteren van geschikte regels wordt nog moeilijker als de condities in een regel niet een exacte toestand beschrijven maar eigenschappen die een toestand moet hebben. Meestal zorgt deze constructie ervoor dat er, alleen al om te bepalen of een toestand voldoet aan bepaalde condities, een complex zoekproces doorlopen moet worden [Rich, 91]. Wanneer het een enkele conditie betreft die gematcht moet worden op een enkel element uit een bepaalde toestand dan kan gebruik worden gemaakt van *unificatie*. Een uitgebreide beschrijving van een unificatie algoritme kan de lezer vinden in [Rich, 91] of [Luger, 89]. Simpele backward-chaining en forward-chaining systemen kunnen eventueel gewoon de eerste toepasbare regel gebruiken. Meestal zullen regel-gebaseerde systemen complexere matching vereisen, al was het alleen maar omdat alle regels die geschikt zijn gevonden moeten worden. Complexere systemen voeren dan conflictresolutie uit met de gevonden regelset. Theoretisch is het mogelijk om het unificatie algoritme voor alle regels en alle condities uit te voeren maar deze methode is hopeloos inefficiënt. Beter is het om het probleem te zien als een many-to-many (veel op veel) matching probleem [Rich, 91]. Het RETE-match algoritme is een efficiënt algoritme dat gebruik maakt van drie belangrijke kenmerken [Rich, 91]:

- Het tijdelijke karakter van data. Regels zullen over het algemeen maar enkele eigenschappen van een toestand wijzigen. Het grootste deel van een toestandsbeschrijving blijft gelijk. Wanneer een regel in de vorige cyclus wel toepasbaar (te controleren) was maar niet uitvoerbaar dan is de kans groot dat dat nog steeds zo is. RETE onderhoudt een netwerk/index van condities in regels en selecteert op basis van veranderingen in de toestand welke regels bekeken moeten worden.

- Structurele overeenkomsten in regels. Sommige regels in een systeem zullen grotendeels hetzelfde zijn op bijvoorbeeld één predikaat in de condities na. RETE slaat de regels zo op dat elke conditie maar één keer uitgevoerd wordt per cyclus. Regels delen hun condities wanneer die gelijk zijn.

- Gebonden variabelen over twee of meer condities. Wanneer aan de afzonderlijke condities in een regel voldaan kan worden betekent dit niet dat de regel uitvoerbaar is. Gedoseerd_voor(Kleurstof, Zetmeel) en Gedoseerd_voor(Trash-catcher, Bindmiddel) zorgen er niet voor dat de regel if Gedoseerd_voor(X,Y) and Gedoseerd_voor(Y,Z) then Gedoseerd_voor(X,Z) kan vuren. RETE onthoudt dit soort afhankelijkheden en kijkt naar aanleiding van veranderingen in de toestand of de regel inmiddels wel toepasbaar is.

Nog complexere matching-methodes zijn nodig wanneer de regels toegepast moeten worden als de precondities *ongeveer* voldoen aan de huidige toestand. Dit is bijvoorbeeld het geval bij spraakherkenning waarbij regels toegepast worden die gesproken tekst omzetten naar phonetisch schrift. Er zullen zoveel verschillen zijn tussen de uitgesproken teksten (door achtergrondgeluiden, individuele spraakverschillen) dat de kans erg klein is dat een regel exact gematcht kan worden. Het werken met "approximate matching " zoals dit proces genoemd wordt, is extra moeilijk omdat de juiste tolerantie gevonden moet worden bij het matchen. Wanneer het systeem te soepel is herkent het matcht te veel, wanneer het niet soepel genoeg is matcht het misschien helemaal niks. Daar komt nog bij dat wanneer het systeem te veel matcht, dus ook het aantal te doorlopen regels toeneemt waardoor de zoektijd van het systeem erg snel kan toenemen. Voor veel taken zijn deze systemen ondanks deze nadelen wel beter geschikt dan de exacte systemen omdat die in bepaalde situaties stoppen met redeneren omdat er geen regels meer gematcht kunnen worden.

Het resultaat van het matchen is een lijst regels die toepasbaar zijn. Vervolgens is het de taak van het zoekmechanisme om te bepalen in welke volgorde de regels toegepast moeten worden. Soms kan het efficiënt zijn om een deel van deze taak ook te laten uit te voeren door het matching proces. Deze stap bij het matchen is de conflictresolutie

7.2.5. Conflictresolutie

Het karakter van een expertsysteem kan aangepast worden door een zogenaamde conflictresolutie-strategie toe te passen bij het matchen. Deze beïnvloedt de volgorde waarin de regels uit de conflictset behandeld worden. Een voorbeeld van zo'n strategie is het sorteren van de conflict set op grond van het aantal condities in een regel waaraan niet reeds voldaan is. In deze paragraaf komen drie algemene methodes van conflictresolutie aan bod [Rich, 91].

Voorkeur op basis van de regels zelf

Er zijn twee veel voorkomende manieren om voorkeur te bepalen op basis van de regels zelf. De gemakkelijkste is op basis van de volgorde waarin de regels gespecificeerd zijn. De hoogste prioriteit geldt voor de eerst gespecificeerde regels. Prolog bijvoorbeeld werkt op dergelijke manier. De predikaten die voldoen worden geprobeerd in de fysieke volgorde waarin ze in het programma voorkomen.

Een twee manier is op basis van de algemeenheid van een regel. Hoe specifieker een regel, hoe kleiner de resterende zoekruimte als zo'n regel toegepast kan worden. Dus is het efficiënt om de specifiekere regels de hoogste prioriteit te geven. De matcher kan beslissen om een regel die algemener is dan een eerder gevonden (ook toepasbare) regel te verwerpen. Er zijn twee makkelijke manieren te noemen om te bepalen welke regel specifieker is dan een ander:

- Als de verzameling precondities van regel A alle precondities van regel B bevat (en meer), dan is regel A specifieker dan regel B.
- Als de precondities van regel A en B gelijk zijn, maar in de precondities van B worden variabelen gebruikt waar bij regel A constanten gebruikt worden, dan is regel A specifieker.

Voorkeur op basis van de objecten

Een andere manier waarmee de voorkeur voor bepaalde regels bepaald kan worden is door de keuze van de regels af te laten hangen van het belang van bepaalde objecten die in die regel voorkomen. Dit kan op verschillende manieren gebeuren. In het bekende ELIZA [Weizenbaum, 66] programma, dat het gedrag van een psychiater probeerde te simuleren, werd ook op basis van de objecten een keuz gemaakt. ELIZA zocht naar

overeenkomsten in de invoer-regel van de gebruiker en de precondities van de regels om antwoorden te genereren:

(X me Y)	→	(X you Y)
(I remember X)	→	(Why do you remember X just now?)
(My {family-member} is Y)	→	(Who else in your family is Y?)
(X {family-member} Y)	→	(Tell me more about yout family)

Vaak voldeed een invoer-regel aan meerdere patronen zodat er meerdere regels toegepast zouden kunnen worden. Het matching proces zocht de regel op met de objecten met de hoogste prioriteit. De invoer-regel "Everybody laughs at me" kan door ELIZA beantwoord worden met "Everybody laughs at you". Maar ELIZA bezat ook een regel om het woord "everybody" te herkennen. Omdat "everybody" minder vaak voorkomt dan "me" en daardoor als belangrijker gerangschikt antwoord het systeem met "who in particular are you thinking of?".

In [Lucas, 88] wordt een vaak toegepaste methode van conflictresolutie beschreven met behulp van *time-tags*. Elk feit in de gegevensverzameling wordt voorzien van een time-tag variabele die het feit op unieke wijze identificeert en tevens een volgorde van recentheid (en daarmee prioriteit) aanduid. Omdat een variabele op meerdere tijdstippen een waarde kan krijgen is het mogelijk dat een regel meerdere m alen in de conflictset kan voorkomen. In een bepaalde situatie kunnen bijvoorbeeld de volgende feiten in de feitenverzameling voorkomen:

tag_1: x = 12
tag_2: x = 8

Een regel zoals if greaterthan(x, 6) then assign(x, 6) komt dan twee keer in de conflictset voor. Om dit probleem op te lossen worden niet de produktieregels maar regelinstantiaties (rule-instances) opgenomen in de conflictset. Een regelinstantiatie is een paar (R, f) waarin R een regel is en f een verzameling feiten zodanig dat elk van de condities van de regel R matcht met een element van f en elk element van f matcht met een conditie van R [Lucas, 88]. Aan de hand van de time-tag wordt dan de instantiatie met het tweede feit geselecteerd omdat die een conditie bevat waarin het meest recente feit gebruikt wordt.

Voorkeur op basis van toestanden

Een derde mogelijkheid is het beoordelen van resultaattoestanden van de toepasbare regels. Alle regels worden uitgevoerd in een tijdelijke omgeving en aan de hand van een heuristische functie wordt de regel die de beste toestand oplevert uitgekozen. Deze methode is gebaseerd op de algemene *best-first zoekmethode*. Het is mogelijk om dit als conflictresolutie-strategie te gebruiken bij het matchen maar over het algemeen wordt het gebruikt als control-knowledge.

Control-knowledge is kennis die opgeslagen is in het systeem met eigenlijk hetzelfde doel als de conflict-resolutie strategieen; het beperken van de te doorlopen zoekruimten. Control-knowledge bevat kennis over welke zoekpaden het programma het eerst moet doorlopen om een zo snel en goed mogelijk resultaat te krijgen. Een aantal veel gebruikte vormen van control-knowledge zijn:

- Kennis over welke toestanden de voorkeur genieten
- Kennis over welke regels in welke toestanden toegepast kunnen worden
- Kennis over de volgorde waarin het best aan de subdoelen voldaan kan worden
- Kennis over efficiënte volgorde voor het toepassen van regels

De tweede en vierde methode worden in de ontwikkelomgeving G2 standaard ondersteund. Hierover meer in hoofdstuk 9.
Voor een verdere uitwerking van de problemen en aspecten bij bottom-up en top-down inferentie in produktiesystemen wordt de lezer verwezen naar [Lucas, 88]. Hierin zijn tevens voorbeelden opgenomen voor de implementatie van het systeem in PROLOG en LISP.

7.4. Onzekerheid bij het redeneren

Alle methodes die tot nu toe besproken zijn in dit hoodstuk werken met feiten die waar zijn, niet waar of niet bekend. Voor een bepaald soort problemen kan het noodzakelijk zijn ook feiten te representeren die niet niet zeker zijn maar waarvoor ondersteunend bewijs bestaat. Grofweg zijn deze problemen onder te verdelen in twee soorten. Bij de eerste soort behoren bijvoorbeeld de kansspelen, in het probleemdomein speelt volledige willekeurigheid een rol. Bij het kaarten weten we bijvoorbeeld niet zeker dat de tegenstander bepaalde kaarten wel of niet heeft maar we kunnen wel de statistische kans bepalen. Bij de tweede soort is het geen kwestie van willekeurigheid. Het gaat hier om kennis waarvan we kunnen zeggen dat het in de meeste gevallen waar is, tenzij het om een uitzonderingssituatie gaat. Het grootste probleem bij dit soort problemen is dat er over het algemeen meer uitzonderingen zijn dan we expliciet willen opnemen in de kennis. Ook bij dit soort problemen kunnen statistische waarden een belangrijk hulpmiddel zijn om te representeren hoe groot de kans is dat een uitzondering plaats vind. In deze paragraaf zullen een aantal technieken behandeld worden om de mate van onzekerheid of zekerheid in een expertsysteem te verwerken:

- Zekerheidsfactoren
- Bayesiaanse netwerken
- Dempster-Shafer theorie

De drie van bovengenoemde methoden zijn gebaseerd op Bayes theorie. Daarom wordt eerst de basis van die theorie uitgelegd. De in paragraaf 6.1.3. beschreven fuzzy logic is ook een mogelijk gereedschap om onzekerheid te representeren, die overigens in beperkte mate standaard door G2 ondersteund wordt.

Het SIX prototype redeneert niet met onzekerheid. Uit de kennis die tijdens interviews (paragraaf 10.1.) naar voren kwam bleek dat dit niet noodzakelijk was voor de benodigde functionaliteit. Omdat een deel van de doelstelling het voor Vertis vergaren van kennis over expertsystemen was wordt er in dit hoofdstuk wel aandacht aan besteed. Verder is het denkbaar dat in de toekomst wanneer de expertkennis verder uitgediept wordt of extra kennis kan worden afgeleid uit een model het voor het redeneren wel noodzakelijk is rekening te houden met onzekerheid.

Waarschijnlijkheid en Bayes theorie

De basis van Bayes theorie is de conditionele waarschijnlijkheid: $P(H|E)$
Dit is de weergave van de kans P dat een gebeurtenis of hypothese H optreedt wanneer een gebeurtenis of feit E bekend is. Bij medische diagnose is het bijvoorbeeld bekend hoe groot de statistische kans is dat een iemand een bepaalde ziekte heeft. Wanneer de patient bepaalde symptonen (E) heeft is de waarschijnlijk (P) groter (of kleiner) dan de statistische kans dat deze patient die ziekte heeft (H). Willen we deze waarschijnlijkheid kunnen berekenen dan moet er rekening gehouden worden met de statistische kans op H (de kans zonder extra bewijzen) en in hoeverre E een bewijs is voor H. Om dit kunnen doen definieert Bayes een universum dat een eindig aantal disjunctie hypotheses H_i bevat. Verder geldt dat:

$P(H_i|E)$ = de kans dat hypothese H_i geldt, gegeven E
$P(E|H_i)$ = de kans dat bewijs E wordt waargenomen als hypothese H_i waar is
$P(H_i)$ = de a priori kans dat hypothese H_i waar is zonder specifieke bewijzen
k = het aantal mogelijke hypotheses.

Bayes theorie stelt dan dat [Rich, 91]:

$$P(H_i|E) = \frac{P(E|H_i) \cdot P(H_i)}{\sum_{n=1}^{k} P(E|H_n) \cdot P(H_n)}$$

Belangrijk voor de werking van deze regel is de exacte betekenis van de term P(A|B). We geven hiermee de conditionele waarschijnlijkheid aan voor A, gegeven dat het enige bewijs is B. Wanneer er nog andere relevante bewijzen zijn moeten die op een bepaalde manier meegenomen worden in de kansbepaling wil deze nog de juiste betekenis houden. Aan de hand van een voorbeeld uit een medisch diagnose systeem zal dit duidelijker worden [Rich, 91]:

A: De patient heeft vlekken
B: De patient heeft de mazelen
C: De patient heeft hoge koorts

Wanneer er verder geen bewijs is, is het duidelijk dat de aanwezigheid van vlekken (A) als bewijs dient voor B (het hebben van de mazelen). Het hebben van vlekken dient in bepaalde mate ook als bewijs voor het hebben van hoge koorts, mazelen gaat immers vaak gepaard met hoge koorts. Maar stel dat we al weten dat de patient de mazelen heeft (B is bewijs). Het extra bewijs A levert dan geen verhoogde waarschijnlijkheid op voor stelling C. Tevens is het zo dat alleen vlekken of alleen koorts een bewijs vormt voor de mazelen. Hierbij is het wel zo dat het ene bewijs het andere kan versterken en daarmee moeten we dus rekening houden als beide gegevens (A & C) bekend zijn. Omdat A en C niet onafhankelijk van elkaar zijn kunnen we niet gewoon de som van hun effecten nemen. De conjunctie van de effecten van A en C moet verwerkt worden in de conditionele waarschijnlijkheid van B. In het algemeen kan gezegd worden,gegeven een eerder verkregen bewijs *e* en een nieuw geobserveerd bewijs E dan

$$P(H|E,e) = P(H|E) \cdot \frac{P(e|E,H)}{P(e|E)}$$

Wanneer we uitgaan van een redelijk complexe wereld, complex genoeg om functioneel te zijn binnen een expertsysteem, dan loopt het aantal gecombineerde waarschijnlijkheden die we berekenen moeten om een conditionele kans te bepalen al snel op tot 2^n (waarbij *n* het aantal verschillende proposities is waarmee we rekening moeten houden [Rich, 91]). Vanwege drie redenen blijkt dan dat het gebruik van de pure theorie in een systeem niet haalbaar is:

- Er moeten te veel waarschijnlijkheden door kennis-acquisitie aangeleverd worden.
- Er zou te veel ruimte nodig zijn om alle waarschijnlijkheden op te slaan.
- Er zou te veel tijd nodig zijn om de waarschijnlijkheden te berekenen.

Ondanks deze praktische problemen van de Bayesiaanse methode biedt het toch een aantrekkelijke basis voor het representeren van onzekerheid in expertsystemen. Er zijn een aantal methoden ontwikkeld die de problemen omzeilen maar wel van de voordelen gebruik maken. Zoals al gezegd in het begin van deze paragraaf komen deze in de rest van dit hoofdstuk aan bod.

7.4.1. Zekerheidsfactoren

Voor het omgaan met onzekerheden in MYCIN is door E.H. Shortliffe en B.G. Buchanan het certainty factor model ontwikkeld. De ontwikkeling van dit model is gemotiveerd vanuit de problemen die zich voordeden bij het toepassen van de waarschijnlijkheidsrekening en bayesiaanse statistiek mee zich mee brengt. In paragraaf 5.2.1. is een voorbeeld opgenomen van een regel uit MYCIN met de bijbehorende representatie. Bij de ontwikkeling van MYCIN bleek dat een expert, wanneer hij de kans P(H|E) = x schatte, het moeilijk te accepteen vond dat P(¬H|E) = 1 - x omdat de gegevens die gebruikt werden als bevestiging van de juistheid H niet tevens kunnen dienen als bevestiging van de onjuistheid van H. Hieruit werd de conclusie getrokken dat de expert bij de regels niet echt de kans schat dat H juist is, maar meer een maat van vertrouwen in wantrouwen aan een de beschouwde hypothese. In het certainty factor model worden daarom twee basismaten van onzekerheid onderscheiden: de mate waarin bepaalde gegevens een hypothese bevestigen (*measure of belief, MB* een waarde tussen 0 en 1) en de mate waarin bepaalde gegevens een hypothese juist verwerpen (measure of disbelief, MD een waarde tussen 0 en 1). Uit deze twee, door de expert geleverde waarden, wordt een zekerheidsfactor (CF) voor een bepaalde regel berekend:

$$CF[h,e] = MB[h,e] - MD[h,e]$$

Omdat elk gegeven of een bewijs voor (MD = 0) of een bewijs tegen (MB = 0) een bepaalde hypothese is, en elke MYCIN regel overeenkomt met een enkel gegeven dat als bewijs kan dienen is één getal bij elke regel voldoende om MB en MD en dus CF te definieeren. Om tijdens het redeneren gebruik te kunnen maken van deze waarden moeten de waarden op drie manieren gecombineerd kunnen worden:

1. Twee regels leveren bewijs voor één hypothese en de CF's van die regels moeten op een bepaalde manier gecombineerd worden ($A \rightarrow C$, $B \rightarrow C$).
2. Van verschillende condities in één regel moet de mate van vertrouwen gecombineerd kunnen worden om te kijken ($A \wedge B$, $A \vee B$).
3. De uitvoer van de ene regel moet als invoer dienen voor een andere regel ($A \rightarrow B \rightarrow C$).

Voor het eerste geval geldt in het certainty factor model de volgende regel:

$$MB[h, s_1 \wedge s_2] = \begin{cases} 0, \text{ voor } MD[h, s_1 \wedge s_2] = 1 \\ MB[h, s_1] + MB[h, s_2] \cdot (1 - MB[h, s_1]) \end{cases}$$

$$MD[h, s_1 \wedge s_2] = \begin{cases} 0, \text{ voor } MB[h, s_1 \wedge s_2] = 1 \\ MD[h, s_1] + MD[h, s_2] \cdot (1 - MD[h, s_1]) \end{cases}$$

Dit wil zeggen dat de mate van waarschijnlijkheid voor H 0 is als het zeker is dat H niet waar is. Anders is de mate van waarschijnlijkheid gelijk aan de mate van waarschijnlijkheid van de eerste waarneming (MB[h,s_1]) vermeerderd met een bepaalde hoeveelheid voor de mate van waarschijnlijkheid van de tweede waarneming (MB[h,s_2]). Deze hoeveelheid wordt bepaald door het verschil te nemen van MB[h,s_1] en 1 (want dat is het maximum dat de waarde verhoogd zou kunnen worden) en dat te vermenigvuldigen met MB[h,s_2] (zodat deze geschaald wordt op de maximale verhoging). Hetzelfde geldt voor MD. Op deze wijze wordt bij het combineren van factoren op de juiste wijze CF verhoogd danwel verlaagd.

De formules die MYCIN gebruikt voor het berekenen van MB in het tweede geval, de disjunctie respectievelijk conjunctie van twee hypotheses is als volgt (voor berekening van MD gelden dezelfde formules):

$$MB[h_1 \wedge h_2, e] = \min(MB[h_1, e], MB[h_2, e])$$

$$MB[h_1 \vee h_2, e] = \max(MB[h_1, e], MB[h_2, e])$$

Ten slotte moet het derde geval nog bepaald kunnen worden. Wanneer we het resultaat van een regel verder willen gebruiken moet rekening gehouden worden de zekerheidsfactor van de gebruikte regel en met de mate van vertrouwen in het bewijs dat we voor die regel gebruikt hebben. Wanneer MB'[h,s] de mate van vertrouwen is in h, wanneer we absoluut zeker zijn van s (de CF van een regel) en e het bewijs waarvan s is afgeleid dan geldt:

$$MB[h,s] = MB'[h,s] \cdot \max(0, CF[s,e])$$

Het is voor het gebruik van de hierboven beschreven methode niet nodig om MB en MD verder te specificeren. De originele auteurs deden dat wel[5] en met enige kleine aanpassingen zijn de definities van deze twee waarden zo te noteren dat de basis van de methode overeenkomt met die gebruikt voor pure bayesiaanse statistiek [Rich, 91]. Hoe kan het dat de MYCIN methode werkt en de Bayes theorie in pure vorm als niet praktisch bruikbaar is? De bayesiaanse theorie definieert de conditionele kans P(H|E) als E het enige waargenomen relevante bewijs is, wanneer er meerdere relevante bewijzen waren moesten deze gecombineerd worden en dat was waar de problemen ontstonden. MYCIN negeert dit probleem min of meer. De hierboven gegeven formules gaan er van uit dat de regels onderling onafhankelijk zijn van elkaar. De verantwoordelijkheid voor deze onafhankelijkheid ligt bij diegene die de regels opsteld. Alle drie de formules (en het normale gedrag van het systeem) zijn kwetsbaar zogauw de regels niet voldoen aan deze onafhankelijkheidsaanname.

Uit een simpel en duidelijk voorbeeld [Rich, 91] blijkt al snel hoe kwetsbaar.

S: Sproeier was gisteravond aan.
N: Het gras is nat vanochtend.
R: Gisteravond regende het.

[5] Zie voor een uitgebreide beschrijving van het certainty factor model op basis van de bayesiaanse theorie [Lucas, 88] of het originele werk van [Buchanan en Shortliffe, 84].

Met behulp van MYCIN-achtige regels zouden we dit als volgt kunnen representeren:

1. als: Sproeier was gisteravond aan
   dan is er bewijs (0.9) dat het gras vanochtend nat is.

2. als: Het gras is nat vanochtend
   dan is er bewijs (0.8) dat het gisteravond regende.

Beide regels apart beschouwd lijken in orde. Als de sproeier gebruikt is, is het erg waarschijnlijk dat het gras nat is. En normaal gesproken is het erg waarschijnlijk dat een nat grasveld duidt op een regenbui kort geleden. Als bekend is dat er gesproeid is en we combineren de regels volgens de MYCIN formules is het resultaat:

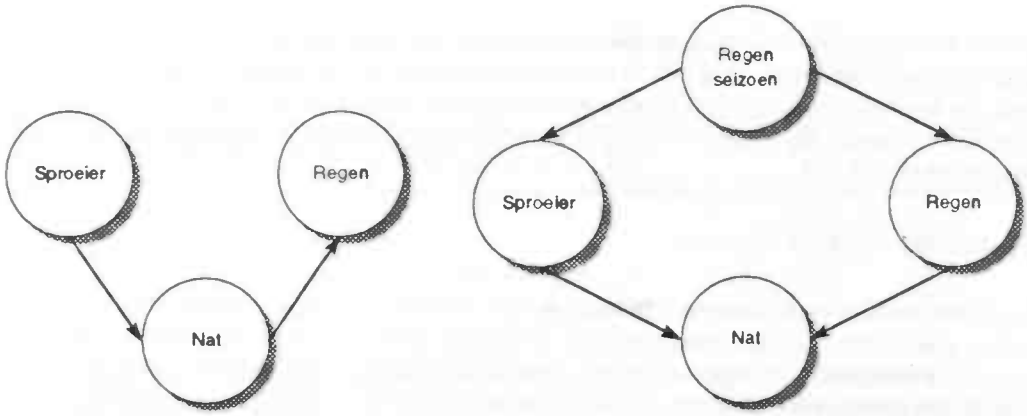
$$MB[W,S] = 0{,}8$$
$$MB[R,W] = 0{,}8 \cdot 0.9 = 0.72$$

Met andere woorden we geloven dat het geregend heeft omdat de sproeier aangestaan heeft, terwijl (of dankzij dat) er bewijs is dat de sproeier gezorgd heeft voor een nat grasveld[6]. Het voordeel van onafhankelijke (modulaire in te voeren) regels is tevens het grootste gevaar voor de methode. Het erg belangrijk dat er nagedacht is over de regels die ingevoerd worden om ongewenste effecten en conclusies te voorkomen. Ondanks dit nadeel van de methode kan het certainty factor model in praktijk heel goed als basis dienen voor een expertsysteem. Tijdens de ontwikkeling van MYCIN [Buchanan, 84] en een groot aantal systemen ontwikkeld met EMYCIN, een shell gebaseerd op MYCIN, is dit gebleken. Voor een voorbeeld van implementatie van de methode wordt de lezer verwezen naar [Lucas, 88] waarin het principe vrij uitgebreid behandeld is en aan de hand van EXSHELL, een in PROLOG ontwikkelde shell gebaseerd op certainty factors, de werking van een simpele applicatie duidelijk beschreven is.

7.4.2. Bayesiaanse netwerken

Een andere aanpak ontwikkeld om met onzekerheid te redeneren is met behulp van Bayesiaanse netwerken ([Pearl, 88], [Rich, 91]). Deze methode handhaaft de originele Bayesiaanse aanpak maar gaat daarbij uit van de modulariteit van de wereld waarover geredeneerd moet worden. Wanneer we (een deel van) de wereld willen representeren is het niet nodig om een gigantisch grote tabel bij te houden waarin alle waarschijnlijkheden voor alle mogelijke combinaties van gebeurtenissen (gegevens). Het grootste deel van die gebeurtenissen zal toch conditioneel onafhankelijk van elkaar zijn dus is het niet nodig die waarschijnlijkheden op te slaan. Er kan dus gebruik gemaakt worden van een soort lokale representatie waarin steeds per groep of cluster van gebeurtenissen de afhankelijkheden beschreven worden. In een aantal gebieden van de kunstmatige intelligentie wordt gebruik gemaakt van (grafisch weergegeven) netwerken om bepaalde kennis te representeren. Ook bij Bayesiaanse netwerken, zoals de naam al doet vermoeden, speelt dit een belangrijke rol. In de figuur 7.4. (a) is het voorbeeld uit de vorige paragraaf nog eens weergegeven met behulp van een netwerk. De pijlen representeren de richting waarin de regels opgesteld waren.

[6] In dit specifieke geval zou het probleem opgelost zijn als beide regels als causale verbanden beschreven zouden zijn. De eerste regel beschrijft een causaal verband (de sproeier veoorzaakt het natte gras). De tweede regel echter is invers causaal (het natte gras is veroorzaakt door de regen en dus bewijs voor de oorzaak). Conclusies die causaal zijn afgeleid mogen niet gebruikt worden om invers causaal iets af te leiden. Dit is probleem waar veel methoden mee kampen [Rich, 91].



(a) (b)

Figuur 7.4. Netwerkrepresentaties van causaliteit

In dit voorbeeld kwam een probleem naar voren met de causaliteit van de regels. Het eigenlijke probleem bestaat er uit dat er geen onderscheid is gemaakt in het soort invloed van proposities. Een gegeven kan op twee manieren invloed uit oefenen. Oorzaken hebben invloed op de waarschijnlijkheid van hun symptomen en de waarneming van een symptoom beïnvloedt the waarschijnlijkheid van al zijn oorzaken. Bayesiaanse netwerken zijn gebaseerd op een duidelijk onderscheid tussen deze twee mogelijke invloeden.

Door middel van een gerichte niet-cyclische graaf ("directed acyclic graph" of DAG) worden causaliteits relaties tussen variabelen gerepresenteerd. In figuur 7.4. (b) is het eerder genoemde voorbeeld door middel van een DAG weergegeven waarin de pijlen de causaliteitsrelaties weergeven. Aan het voorbeeld is een variabele toegevoegd die weergeeft om het momenteel een regenachtig seizoen is.. Om deze graaf als basis voor redeneren met behulp van waarschijnlijkheden te kunnen gebruiken moet er nog meer informatie beschikbaar zijn. Van elke waarde van een ouder knoop moet bekend zijn in hoeverre ze een bewijs vormen voor de waarden die de kinder-knopen kunnen aannemen. In de onderstaande voorbeeld tabel zijn de conditionele waarschijnlijkheids waarden opgenomen voor het voorbeeld [Rich, 91].

Gebeurtenis	*Waarschijnlijkheid*
p(Nat|Sproeier, Regen)	0.95
p(Nat|Sproeier, ¬Regen)	0.9
p(Nat|¬Sproeier, Regen)	0.8
p(Nat|¬Sproeier, ¬Regen)	0.1
p(Sproeier|Regen seizoen)	0.0
p(Sproeier|¬Regen seizoen)	1.0
p(Regen|Regen seizoen)	0.9
p(Regen|¬Regen seizoen)	0.1
p(Regen seizoen)	0.5

Tabel 7.1. Conditionele waarschijnlijkheden voor een Bayesiaans netwerk

Uit deze tabel kan bijvoorbeeld opgemaakt worden dat de (onafhankelijke) kans op een dag in een regenachtig seizoen 0.5 is. Vervolgens als het bekend is dat het een dag in een regenachtig seizoen is, is de kans op regen op die dag 0.9. In de andere seizoenen (¬Regen seizoen) is diezelfde kans op regen slechts 0.1.

Deze methode is alleen praktisch toepasbaar in expertsystemen als er een mechanisme beschikbaar is waarmee de invloed van een willekeurige knoop op een willekeurig andere knoop berekent kan worden. Wanneer bijvoorbeeld bekend is dat het regent, hoe groot is de kans dat het een dag in een regenachtig seizoen is? Om dit te kunnen bepalen moet de DAG geconverteerd worden naar een ongerichte graaf waarmee via de verbindingen in alle richtingen waarschijnlijkheden doorgestuurd kunnen worden. Het mechanisme moet tevens garanderen dat dit op de juiste wijze gebeurt. Ondanks dat het terecht is om te zeggen dat de observatie

van nat gras een bewijs vormt voor regen, en dat de observatie van regen een bewijs vormt voor nat gras moet het mechanisme garanderen dat de graaf niet in een cirkel doorlopen wordt waarin de regen het bewijs vormt voor nat gras, dat weer bewijs vormt voor regen, dat weer bewijs vormt voor nat gras, enz. Voor dit soort mechanismes zijn een aantal algoritmes ontwikkeld waaronder bijvoorbeeld het *message-passing algoritme* zoals beschreven in [Pearl, 88].

7.4.3. Dempster-Shafer theorie

In de zestiger jaren werd door A. Dempster de basis gelegd voor een wiskundige theorie voor onzekerheid. In 1976 werd deze theorie door G. Shafer uitgebreid tot de theorie die bekend is als de Dempster-Shafer theorie. Deze theorie is niet specifiek voor expertsystemen ontwikkeld maar pas sinds de jaren tachtig begint men in te zien dat de methode hiervoor wel bruikbaar kan zijn [Lucas, 88]. Er is inmiddels door verschillende onderzoekers een aantal manieren voorgesteld om de theorie toch te kunnen gebruiken. De theorie is afwijkend van de hier eerder besproken methoden omdat deze theorie geen waarschijnlijkheden toekent aan enkele proposities maar waarschijnlijkheden aan verzamelingen proposities. In deze paragraaf wordt een korte beschrijving gegeven van de basis van deze theorie.

In de theorie van Dempster en Shafer kan er in plaats van met afzondelijke hypotheses met verzamelingen, dat wil zeggen disjuncties, van hypothesen een maat van onzekerheid verbonden worden. Het voordeel hiervan is dat er onderscheid gemaakt kan worden tussen onzekerheid en onwetendheid. De methode om met onzekerheden om te gaan komt kortweg hierop neer: uitgaande van de verzameling van alle hypothesen wordt telkens een maat van onzekerheid met bepaalde deelverzamelingen geassocieerd op basis van aanwijzingen. Dit gaat door totdat er maten van onzekerheid zijn geassocieerd met verzamelingen bestaande uit één element.

Met behulp van een voorbeeld blijkt al snel het verschil met de andere methoden. Stel dat er in het papierfabricageproces vier mogelijke problemen zijn:
A) verkeerd zetmeel gedoseerd
B) overdosering van een kationisch additief
C) zetmeel werkt als trash-catcher
D) ruimte in het proces voor zetmeel wordt niet optimaal gebruikt

De beginverzameling die de verzameling van deze vier afzondelijke hypothesen {A, B, C, D} bevat en alle deelverzamelingen daarvan (ook wel onderscheidingsverzameling Θ genoemd) wordt geassocieerd met waarschijnlijkheid 1. De kans op of A of B of C of D is 1. Voor elke element in Θ geldt dat de waarschijnlijkheid in het interval [0,1] ligt, er is immers nog niks bekend. De andere methoden zouden initieel 0.25 aan iedere hypothese toekennen. Er is dan niet duidelijk of uit alle bewijzen blijkt dat de kans voor alle hypothesen even groot is of dat er nog geen aanwijzingen zijn. Stel nu dat slechte zetmeelretentie wordt geconstateerd en er in het voorbeeld gemakshalve aangenomen wordt dat dit een bewijs (symptoom) met sterkte 0.6 is voor of verkeerd zetmeel (A) of overdosering van kationisch additief (B). De theorie van Dempster-Shafer kent vervolgens de waarde 0.6 toe aan de verzameling {A, B} en blijft er 0.4 over voor Θ. Hiermee wordt dus niet 0.4 toegekent aan het complement van {A, B}. {A, B} is immers ook element van Θ. Wederom zouden de hiervoor besproken methoden de 0.6 verdelen tussen A en B, waardoor het weer lijkt alsof er meer bekend is dan er werkelijk bekend is.

De problemen voor de implementatie van de Dempster-Shafer theorie komen vooral voort uit de computationele complexiteit van theorie (als gevolg van de bijbehorende door Dempster ontwikkelde combinatieregel) en het ontbreken van een aantal combinatiefuncties. Het toepassen van Dempster's combinatieregel is exponentieel omdat gebruik gemaakt wordt van de machtsverzameling van de onderscheidingsverzameling. Voor een uitgebreidere verhandeling en theoretische onderbouwing van deze theorie wordt de lezer verwezen naar [Lucas, 88]. Hierin wordt tevens een methode besproken om de implementatieproblemen te omzeilen.

8. DE USER INTERFACE

R.A. Carton

De laatste jaren wordt er veel aandacht besteed aan één van de belangrijkste onderdelen van computerprogramma's: de user interface. De mystiek die vroeger bij het fenomeen "computer" hoorde, is deels te wijten aan de slechte user interfaces van de toenmalige machines. Om met een computer van pakweg 30 jaar geleden overweg te kunnen, moest de programmeur (een leek kwam er niet aan te pas) beschikken over veel kennis van de machine zelf en het operating system dat erop draaide. Ook nauwkeurigheid was belangrijk, want één fout kon een grote hoeveelheid werk genadeloos doen verdwijnen. Natuurlijk waren niet alle typefouten desastreus. Maar ze kostten wel veel tijd, aangezien ze pas werden geconstateerd als de programmeur de uitvoer van zijn "job" ging ophalen, enkele uren (of soms dagen) nadat hij het programma aan de computer had aangeboden.

Door de technologische vooruitgang van de laatste decennia en de aandacht die wordt besteed aan de ontwikkeling van het "*front-end*" van software is er gelukkig veel veranderd. Vrijwel iedereen kan zich nu een computer veroorloven en de meeste mensen kunnen er ook mee werken. De huidige computers zijn snel en worden veelal door slechts één persoon gebruikt, zodat resultaten niet lang op zich laten wachten. En hoewel het nog steeds mogelijk is om met enkele handelingen grote hoeveelheden informatie te vernietigen [7], wordt de gebruiker meestal gewaarschuwd voor handelingen die "gevaarlijk" zijn en is er de mogelijkheid een opdracht te annuleren.

Dit hoofdstuk behandelt verschillende typen user interfaces met als doel de meest geschikte interface voor SIX te vinden. Na een karakterisering van het probleem dat aan de interface-keuze voor SIX tèn grondslag ligt, volgt een korte algemene inleiding over de communicatie tussen mensen en computers. Daarna worden er enkele soorten user interfaces beschreven. Bij het zoeken naar een goede interface voor het expertsysteem blijkt dat elk van de beschreven typen voor- en nadelen heeft. Op grond van deze voor- en nadelen wordt er vervolgens een keuze voor de gebruikersinterface van SIX gemaakt. Ten slotte vindt er een evaluatie plaats van de gemaakte keuze.

8.1. Voorwaarden voor de interface van SIX

Het expertsysteem SIX moet een voorbeeldsysteem worden. Zoals in hoofdstuk 2 is beschreven, moet AVEBE door SIX een indruk krijgen van expertsystemen die advies geven over zetmeeltoepassingen. De toekomstige gebruikers van SIX (of van een uitgebreidere versie daarvan) zullen medewerkers van het Papierlab van AVEBE zijn, of vertegenwoordigers van AVEBE. Van hen wordt niet noodzakelijkerwijs verwacht dat ze ervaring met computers en expertsystemen hebben. Verder heeft AVEBE geen opdracht tot ontwikkeling van het systeem gegeven, maar ligt het initiatief bij Vertis. SIX moet AVEBE dus overtuigen van de voordelen van dergelijke expertsystemen.

Deze voorwaarden hebben consequenties voor de keuze van de interface voor het systeem. De gebruikers-interface van SIX moet begrijpelijk en overzichtelijk zijn, ofwel: toegankelijk. Deze eis kan in twee concrete eigenschappen voor het systeem worden uitgedrukt, namelijk:

[7] Het vernietigen van een grote hoeveelheid informatie is tegenwoordig zelfs makkelijker dan vroeger vanwege de gebruikersvriendelijke programmatuur (waarmee bijvoorbeeld een directory en al zijn subdirectories tegelijkertijd naar een prullebak kunnen worden "gesleept") en doordat de capaciteit van een gemiddelde harde schijf met enkele ordes van grootte is toegenomen.

1. Eenvoudige bediening en
2. Aantrekkelijk uiterlijk.

Waarbij "eenvoudig" wil zeggen: begrijpelijk en niet ergerlijk simpel. Het uiterlijk is aantrekkelijk als het past bij de belevingswereld van de doelgroep. Deze gewenste eigenschappen zullen in paragraaf 8.3. verder worden uitgesplitst. Eenvoudige bediening zorgt ervoor dat de gebruiker zich het systeem snel eigen kan maken, ook zonder veel computerkennis. Hierdoor is het gebruik niet bedreigend of demotiverend voor beginners. Als het expertsysteem door vertegenwoordigers van AVEBE bij klanten moet worden gebruikt, is het voor klant en vertegenwoordiger prettig als de bediening zo makkelijk mogelijk is. Een aantrekkelijk uiterlijk wekt gemakkelijk de interesse van de toekomstige gebruiker en draagt bij aan een snelle acceptatie van het systeem. Een programma dat eenvoudig in het gebruik is en er interessant en overzichtelijk uitziet, is geschikt om als demonstratiesysteem te dienen. Kortom: de twee genoemde eigenschappen zijn belangrijk omdat ze de toegankelijkheid van het systeem waarborgen. Bij het selecteren van een interface zal steeds met deze eigenschappen rekening worden gehouden.

8.2. Communicatie tussen mens en computer

Voor goede communicatie tussen mensen en computers zijn afspraken nodig. Afspraken over de apparaten waarmee een computer kan worden bediend (de communicatiemiddelen) en afspraken over de manieren waarop een computer kan worden bediend (de user interfaces). In de volgende paragraaf worden bekende vormen van user interfaces besproken en in deze paragraaf komen de communicatiemiddelen aan bod.

De interactie tussen mensen en computers verloopt via hardware. Bij de mens omvat deze hardware de ogen en de oren waarmee informatie van de computer wordt opgevangen en de handen en de mond waarmee de computer van informatie wordt voorzien (respectievelijk de visuele, auditieve en haptieve zintuigen). Om commando's en gegevens tot zich te nemen kan de computer beschikken over een toetsenbord, een muis, een microfoon en een joystick. Voor het verstrekken van informatie aan de gebruiker is de computer minimaal uitgerust met een beeldscherm[8]. Verder kunnen een printer en speakers worden g ebruikt.

Het toetsenbord wordt het meest gebruikt en is het oudste invoerapparaat[9]. Toetsenborden hebben bepaalde toetsindelingen en de bekendste daarvan zijn de QWERTY– en AZERTY-indeling, de Dvorak-indeling en de alfabetische indeling. Ondanks het feit dat de Dvorak-rangschikking 30% sneller werkt dan de QWERTY-rangschikking en dat de alfabetische rangschikking voor beginners het handigst is, hebben de meeste toetsenborden de QWERTY-indeling [Willemse, 88].

Aanwijsapparaten zoals de muis zijn al jarenlang populair als aanvulling op het toetsenbord. Muizen hebben meestal één, twee of drie knoppen; sommige hebben er meer. Het optimale aantal is niet eenduidig vastgelegd. Muizen krijgen een steeds betere ergonomische vormgeving, waardoor ze prettiger in het gebruik zijn dan vroeger. Joysticks worden voornamelijk voor spellen gebruikt, maar voor sommige toepassingen kunnen ze als aanwijs– of stuurapparaat dienen, net als de muis. In plaats van een muis of een joystick kan ook een trackball, een digitizer, een lichtpen of een aanraakscherm aanwezig zijn. Een trackball is te vergelijken met een muis die ondersteboven ligt: het kastje wordt niet over het bureau verplaatst, maar de kogel wordt met de vingers heen en weer bewogen. Een digitizer lijkt op een muis of op een elektronische pen en stelt de gebruiker in staat om bijvoorbeeld coördinaten in te voeren door punten van een ontwerptekening aan te wijzen. Een lichtpen wordt gebruikt om voorwerpen op het scherm aan te wijzen en werkt dus directer dan een muis die een cursor op het scherm bestuurt. Nog directer is het aanraakscherm, waarbij de gebruiker met zijn vinger objecten op het scherm kan aanwijzen. Het aanraakscherm is minder precies dan de lichtpen en kan bovendien zorgen voor vingerafdrukken op het beeldscherm. Beide invoermethoden hebben het nadeel dat de hand geen steun kan vinden bij het aanwijzen, zoals bij het gebruik van de muis of de digitizer.

---

[8] Of met een hard-copy-terminal, waarbij alle in– en uitvoer direct op papier wordt gezet [Willemse, 88]. Deze apparaten zijn echter zo sterk verouderd, dat ze niet meer als reële alternatieven worden gezien.

[9] Voordat het toetsenbord werd gebruikt, moesten computers door middel van schakelaars en plugborden worden bediend. Deze invoerapparaten zijn dus nog ouder, maar worden hier buiten beschouwing gelaten omdat ze vrijwel nergens meer in gebruik zijn.

De microfoon maakt een heel andere manier van gegevens invoeren mogelijk. Met een microfoon kunnen bijvoorbeeld gesproken commando's worden gegeven. Aan de techniek die daarachter schuilgaat moet nog veel worden verbeterd, want het herkennen van gesproken tekst verloopt haast nooit foutloos. De microfoon kan een nuttig invoerapparaat zijn, dat veel tijd bespaart. Om aan te slaan, moet de spraakherkenningssoftware echter net zo betrouwbaar worden als bijvoorbeeld de stuurprogramma's voor muizen.

Beeldschermen waren vroeger klein en sommige konden alleen tekens weergeven. Dat gebeurde in zwart/wit of zwart/groen. Tegenwoordig zijn er grote, platte beeldschermen met hoge *resoluties*, die in staat zijn afbeeldingen scherp en in natuurgetrouwe kleuren weer te geven. Door de hoge verversingsfrequentie en de relatief kleine hoeveelheid straling van de moderne monitoren, wordt langdurig met een computer werken steeds minder vermoeiend. Dit is een zeer gunstige ontwikkeling, omdat het beeldscherm verreweg het belangrijkste uitvoerapparaat van de computer is en dit ook nog lang zal blijven.

Een ander belangrijk uitvoerapparaat is de printer. Printers zijn de laatste jaren ook sterk verbeterd. De oorspronkelijke matrixtechnologie, waarbij tekens als groepen puntjes op het papier werden "gehamerd", werd vervangen door onder andere de ink jet– en de lasertechnologie. Deze manieren van afdrukken maken het mogelijk met hoge snelheid teksten en afbeeldingen op papier te zetten, zowel in zwart/wit als in kleur, met een kwaliteit die vergelijkbaar is met die van drukwerk. De moderne printers zijn bovendien vrijwel geruisloos. Een nadeel van laserprinters boven andere typen printers is de produktie van het schadelijke gas ozon. Bij de nieuwste laserprinters probeert men dit bijprodukt echter zoveel mogelijk te voorkomen.

Het laatstgenoemde uitvoerapparaat is de speaker. De meeste computers hebben in de behuizing, het beeldscherm of het toetsenbord een kleine speaker. Deze werd aanvankelijk alleen gebruikt voor een piepgeluid om de aandacht van de gebruiker te trekken. Tegenwoordig zijn veel computers uitgerust met goede speakers en elektronika voor het voortbrengen van geluid en muziek. De kwaliteit van het geluid dat door deze systemen wordt ge(re)produceerd is te vergelijken met de geluidskwaliteit van een bescheiden stereo-installatie. Door deze ontwikkeling kan de computer de aandacht van de gebruiker trekken en de gebruiker tegelijkertijd van informatie voorzien, door bijvoorbeeld tekst uit te spreken. Als de gebruiker via een microfoon commando's geeft, is dit een logische wijze van communiceren. Het uitspreken van tekst kan beperkt blijven tot het afspelen van opgenomen woorden en zinnen, maar de computer kan ook zelf het geluid voor zinnen genereren. Computers zijn momenteel in staat geschreven teksten in verstaanbaar Engels om te zetten. De ontwikkelingen op het gebied van spraaksynthese zijn bijna net zo ver als die op het gebied van spraakherkenning.

Natuurlijk is niet alle genoemde hardware altijd nodig. Een aanraakscherm is bijvoorbeeld voor sommige toepassingen met veel keuzemenu's handig, maar heeft het nadeel dat de gebruiker steeds naar het scherm moet reiken. Als de ontwerper van een interface echter effectief verschillende in– en uitvoerapparaten benut, komt dit de efficiëntie van de toepassing ten goede. Een mooi voorbeeld hiervan is het multimediale programma CUBRICON, een prototype van een "knowledge-based" interface systeem [Neal, 91]. Dit programma werd geschreven in het kader van het IMMI (Intelligent Multi-Media Interfaces) project, dat gedeeltelijk wordt gesteund door de Amerikaanse defensie. CUBRICON heeft een natuurlijke taal-interface voor getypte tekst en registreert bij het invoeren van tekst tevens de positie van de muiscursor op het scherm, zodat de gebruiker in de tekst kan refereren aan objecten die op dat moment door de muis worden aangewezen. Bovendien is er een invoermodule voor gesproken tekst. De uitvoer van het systeem wordt via verschillende apparaten aan de gebruiker gepresenteerd. Het systeem beschikt over een spraakgenerator, een grafisch kleurenscherm voor plaatjes en grafieken en een monochroom scherm voor tekst en tabellen. Afhankelijk van het soort informatie, de aard van de met de gebruiker gevoerde dialoog en de mate van relevantie van de informatie voor de gebruiker wordt de uitvoerapparatuur benut. Het programma zorgt dat de gegevens op een consistente en geïntegreerde manier worden overgedragen.

8.3. Soorten user interfaces

Het deel van de interface dat door software wordt verzorgd is net zo belangrijk als het deel dat door de hardware wordt geregeld. De schil van software die de hardware afschermt kan zo simpel zijn als een commando-interface, maar kan ook uit een natuurlijke taal-interface, een menu-georiënteerde interface of een grafische user interface bestaan. Vaak worden ook combinaties van interface-typen toegepast. Hieronder worden de bekendste soorten user interfaces besproken. Daarbij wordt voor ieder type interface gelet op de

bruikbaarheid voor SIX. De bediening en het uiterlijk zijn dus belangrijk. Voor het beoordelen van de interfaces op een eenvoudige bediening en een aantrekkelijk uiterlijk worden de volgende vragen gesteld:

Bediening

1. Wie heeft de leiding; het programma of de gebruiker?
2. Hoe worden opdrachten gegeven?
3. Welke terugkoppeling (feedback) geeft de interface?
4. Hoe tolerant is de interface?

Uiterlijk

5. Welk verband is er tussen de mogelijkheden van het programma en de zichtbare elementen?
6. Hoe worden afbeeldingen en kleuren toegepast?
7. Kan de gebruiker zich een voorstelling van de interface maken?

Ad 1 (Wie heeft de leiding?): Voor beginnende gebruikers kan het prettig zijn als het programma de gebruiker stap voor stap naar zijn doel leidt. De gebruiker hoeft dan alleen vragen te beantwoorden en verder weinig te onthouden. Gevorderde gebruikers daarentegen kunnen deze aanpak ervaren als een beperking van hun vrijheid; ze weten wat ze de computer willen laten doen, maar kunnen dit slechts duidelijk maken door een aantal vragen te beantwoorden, waarvan sommige overbodig of triviaal zijn. Beginnende gebruikers worden graag geleid; ervaren gebruikers nemen graag zelf het heft in handen.
Ad 2 (Hoe worden opdrachten gegeven?): User interfaces hebben uiteenlopende manieren om de opdrachten van de gebruiker tot zich te nemen. Voor beginners kan het geven van opdrachten niet eenvoudig genoeg zijn (en is de muis dus ideaal), terwijl gevorderden vaak complexere methoden prefereren (zoals het geven van commando's met parameters). De complexere methoden lijken meer vrijheid en meer controle over het systeem te bieden dan de eenvoudige invoermethoden.
Ad 3 (Hoe is de feedback?): De feedback van een interface is enorm belangrijk. Door goede feedback weet de gebruiker dat het programma op een toetsaanslag of "muisklik" heeft gereageerd, wat het programma aan het doen is (en dus waarop de gebruiker wacht), waarom er iets fout is gegaan, wat er fout is gegaan en hoe een fout kan worden hersteld. Zonder goede feedback moet de gebruiker al zijn acties controleren en weet hij soms niet of de computer intensief aan het rekenen is of dat deze is vastgelopen.
Ad 4 (Hoe is de tolerantie ?): Voor beginnende èn gevorderde gebruikers is tolerantie zeer belangrijk. De gebruiker moet in staat zijn verkeerde opdrachten ongedaan te maken. En ook al is het omkeerbaar maken van acties[10] voornamelijk de taak van het programma zelf en niet van de interface, toch kan de interface eraan bijdragen. Door bijvoorbeeld de mogelijkheid te bieden een opdracht die wordt uitgevoerd te annuleren, of door te zorgen dat destructieve opdrachten niet makkelijk worden verward met veel gebruikte, "ongevaarlijke" opdrachten.
Ad 5 (Hoe is het verband tussen "zien" en "kunnen"?): De interface van een programma moet beginnende gebruikers een goede indruk geven van de mogelijkheden van dat programma. Als de gebruiker aan de interface kan zien wat hij ongeveer met het programma kan doen (of in ieder geval hoe hij meer over het programma te weten kan komen), zal hij de bediening makkelijker leren dan wanneer het uiterlijk van het programma geen enkele hint over de mogelijkheden geeft. Een interface die laat zien wat er mogelijk is, ontlast de gebruiker van het onthouden van de programmafuncties en geeft hem een overzichtelijk beeld van de toepassingsmogelijkheden. De ontwerper heeft dan gezorgd voor een goed verband ("mapping") tussen het uiterlijk en de functies. Net als bij gebruiksvoorwerpen geldt dat door observatie moet kunnen worden afgeleid wat er zoal mogelijk is [Norman, 88].
Ad 6 (Hoe worden afbeeldingen en kleuren toegepast?): Het gebruik van kleur en afbeeldingen kan het uiterlijk van een programma overzichtelijker maken. Door kleur te gebruiken is het makkelijker om gegevens op het scherm te groeperen of met elkaar in verband te brengen. En afbeeldingen hebben vaak meer uitdrukkingskracht dan tekst. Daarnaast is een kleurige, grafische interface visueel aantrekkelijker dan een zwart scherm met witte tekst. Leren werken met een aantrekkelijke interface is makkelijker dan met een visueel onprettige interface. Bovendien laat een aantrekkelijk ogend scherm een betere indruk achter op de beginnende gebruiker.
Ad 7 (Kan de gebruiker zich een voorstelling van de interface maken?): Als de gebruiker een interface kan relateren aan iets anders, dat hij goed kent en begrijpt, kan hij zich gemak kelijk een conceptueel model van die

---

[10] Omkeerbare acties zijn acties die volledig ongedaan gemaakt kunnen worden. Een programma dat omkeerbare acties biedt, zal bijvoorbeeld de gegevens die de gebruiker verwijdert nog een bepaalde tijd bewaren om het verwijderen ongedaan te kunnen maken. Soms kan worden volstaan met het bieden van inverses van bewerkingen.

interface vormen. Een conceptueel model is een geheel van ideeën over een object of een handeling (een abstractie van een fysiek model), waarmee het mogelijk is het resultaat van het gebruik van het object of het uitvoeren van de handeling te voorspellen [Norman, 88]. De interface is een applicatie van het conceptuele model. Hiermee kunnen voorspellingen worden gedaan over de mogelijkheden en het gebruik van het programma. Zo'n model maakt het bovendien eenvoudiger om de werking van de interface te onthouden. Om de gebruiker een bepaalde voorstelling van de interface te geven, wordt vaak een metafoor gebruikt. De interface heeft dan bepaalde overeenkomsten met een object uit het dagelijks leven, zoals een controlepaneel of een bureaublad. Door de (gedeeltelijke) analogie met het bekende object leert de beginnende gebruiker sneller met de interface te werken.

Voor elk van de hierna besproken interfaces wordt eerst een inleiding over het type interface gegeven, waarna de zeven bovenstaande vragen worden beantwoord. Bij elk antwoord staat tussen haakjes het nummer van de vraag vermeld. Iedere bespreking eindigt met een evaluatie omtrent de geschiktheid van de interface voor het onderhavige expertsysteem. Het soort interface dat bij de vergelijking van bediening en uiterlijk het best aan de voorwaarden voor SIX blijkt te voldoen, zal gekozen worden voor implementatie.

8.3.1. Commando-interface

De commando-interface is het oudste soort user interface. Het intikken van commando's met parameters vergt veel van de gebruiker en weinig van de computer. De gebruiker moet kennis van de interface hebben, omdat commando-georiënteerde interfaces weinig of geen informatie geven over de mogelijkheden van het pakket. Commando-interfaces zijn voor beginners niet uitnodigend, maar hebben als voordeel dat ze op een eenvoudige manier weergeven wat het systeem doet en gedaan heeft. De gevorderde gebruiker weet welke mogelijkheden er zijn en zal een commando-interface vaak zien als een snelle manier van werken, waarbij hij de handen niet van het toetsenbord hoeft te halen om de muis te bedienen. Omdat bij commando-interfaces weinig visuele informatie wordt gegeven, kan worden volstaan met een klein deel van het scherm voor het geven van opdrachten. De rest van het scherm is dan vrij voor andere zaken [McGraw, 92].

De antwoorden op de bovengenoemde vragen luiden: (1) Bij een commando-interface heeft de gebruiker de leiding. Hij geeft een opdracht, die vervolgens door het programma wordt uitgevoerd. (2) Het geven van een opdracht gebeurt door een commando in te typen, meestal gevolgd door één of meer parameters. (3) De feedback van commando-gestuurde programmatuur laat vaak te wensen over. Veel programma's geven tijdens het rekenen door middel van een regel tekst aan dat ze bezig zijn. Als dat echter niet gebeurt, weet de gebrui ker pas na het beëindigen van het rekenproces dat de computer niet is vastgelopen. (4) Commando-georiënteerde interfaces zijn weinig tolerant. Fouten worden makkelijk gemaakt omdat destructieve en niet-destructieve opdrachten op dezelfde manier worden gegeven, namelijk door middel van commando's. De parameters zijn hierbij vaak doorslaggevend; vergelijk bijvoorbeeld de MS-DOS commando's "FORMAT A:" en "FORMAT C:"; het eerste formatteert een diskette en wordt vaak gebruikt, terwijl het tweede de harde schijf formatteert en vrijwel nooit wordt gebruikt. Het verschil is slechts één letter. Er wordt bij potentieel gevaarlijke opdrachten (zoals het formatteren van de harde schijf in plaats van een diskette) wel een controle uitgevoerd, maar dat helpt niet: het programma vraagt namelijk direct na het intikken van de opdracht om bevestiging. De gebruiker (die een diskette wil formatteren) zal op dat moment tevreden zijn met de actie "formatteren" en deze doorzetten. Hij bevestigt dan dus het commando en niet de parameters [Norman, 88]. (5) Een commando-interface geeft geen enkele hint over de mogelijkheden van het programma. Het scherm geeft over het algemeen erg weinig informatie met betrekking tot het gebruik. (6) Van afbeeldingen is meestal geen sprake en kleuren worden soms toegepast om ergens de aandacht op te vestigen. Dit type interface oogt saai. (7) Veel verder dan "een machine om opdrachten te geven" gaat de voorstelling van de gebruiker niet. Het intikken van commando's is niet te vergelijken met iets anders uit het dagelijks leven.

Een expertsysteem moet een bepaalde hoeveelheid informatie krijgen voordat het kan gaan redeneren. Als de gebruiker alle benodigde gegevens met commando's moet invoeren, is de kans groot dat hij bepaalde zaken vergeet. Een goed systeem zal dan alsnog om het ontbrekende deel vragen. Maar voor de beginnende gebruiker is de commando-interface te passief. Bovendien spreekt deze letterlijk en figuurlijk niet tot de verbeelding. De interface verwacht opdrachten die de gebruiker moet kennen, is niet behulpzaam als de gebruiker de weg kwijt is en kan onverbiddelijk zijn als er een fout wordt gemaakt. De bediening is dus niet zo eenvoudig en van een aantrekkelijke interface is geen sprake. Een commando-interface is daarom voor de beginner niet de meest geschikte keuze.

8.3.2. Natuurlijke taal-interface

Natuurlijke taal-interfaces vormen een heel andere categorie. Ze stellen de gebruiker in staat de computer in natuurlijke taal (meestal Engels) opdrachten te geven (vaak gaat het hier echter om uitgebreide commando-interfaces). Voorlopig is hiervoor nog een toetsenbord nodig, maar naarmate de spraakherkenning door computers beter wordt, komt de mogelijkheid om normaal tegen een computer te praten dichterbij. Het voordeel van een natuurlijke taal-interface is dat de gebruiker geen commando's met parameters hoeft te kennen om met het programma te werken; kennis van de Engelse taal is in het ideale geval voldoende. In werk elijkheid is het echter nog steeds nodig een idee van de woordenschat van het systeem te hebben en te weten welk soort zinnen door de interface worden geaccepteerd. En zelfs met deze kennis kunnen er opdrachten worden gegeven die het systeem verkeerd interpreteert. Bij communicatie tussen mensen is er vaak sprake van dubbelzinnigheid. Het kan de computer moeite kosten om semantisch dubbelzinnige verwijzingen op te lossen, zoals het woord "hij" in de volgende zin:

"De leraar liet Jan nablijven nadat *hij* had afgekeken."

Vanwege onze kennis van de wereld is dit voor mensen nauwelijks dubbelzinnig te noemen. Maar naar wie zijn de verwijzingen "hij" en "hem" in de onderstaande zin?

"De leraar liet Jan nablijven nadat *hij* tegen *hem* had geschreeuwd."

Om deze vraag te beantwoorden is de context van de zin nodig. Een voorbeeld van een ander soort dubbelzinnigheid, namelijk syntactische dubbelzinnigheid is deze opdracht die aan een expertsysteem werd gegeven [McGraw, 92]:

"Show me the ships on a training mission."

Het systeem reageerde met:

"Please define a ME."

Het is duidelijk dat computers moeilijkheden kunnen ondervinden met zinnen die voor mensen eenvoudig zijn. Het automatisch herkennen van natuurlijke taal verloopt nog verre van perfect. Iets dat ogenschijnlijk makkelijker is, het *genereren* van natuurlijke taal, levert echter ook nog problemen op. De natuurlijke taal-interface heeft meestal een natuurlijke taal-generator waarmee antwoorden en boodschappen worden getoond of uitgesproken. De uitvoer van systemen met zo'n generator is beter dan die van de commando-georiënteerde systemen met hun korte, vaak cryptische meldingen. En natuurlijke uitvoer past ook in het geheel: als de vraag in gewoon Engels wordt gesteld, dan moet het antwoord in gewoon Engels worden gegeven. Maar waar mensen zonder veel moeite bepaalde stukken informatie kunnen samenvoegen tot een antwoord dat in de context van een bepaald gesprek past, laten computers het nog vaak afweten. Het programma moet beslissen wanneer het iets moet overbrengen, wat het moet zeggen en hoe het de informatie moet structureren zodat het voor de gebruiker begrijpelijk is. En als de informatie complex is, bijvoorbeeld doordat verschillende regels verschillende groepen gegevens aanleveren, zijn deze beslissingen moeilijk [McGraw, 92].

De zeven vragen kunnen als volgt worden beantwoord: (1) Net als bij de commando-interface heeft de gebruiker de leiding. (2) De opdrachten worden ingetikt en zijn minder cryptisch dan commando's met parameters. (3) Aangezien het systeem meestal in natuurlijke taal antwoordt, zijn de antwoorden begrijpelijk. Ze kunnen echter ook dubbelzinnig zijn. Tijdens het afleiden van de antwoorden door het systeem is de voortgang voor de gebruiker echter niet altijd goed waarneembaar. (4) Door het gebruik van natuurlijke taal is de kans op het door elkaar halen van opdrachten kleiner dan bij de commando-interface. De dubbelzinnigheid van natuurlijke taal kan tot gevolg hebben dat de gebruiker zijn opdrachten vaak moet bevestigen. Dit gebeurt dus niet zozeer om de gebruiker voor fouten te behoeden, maar meer om interpretatiefouten van de interface te voorkomen. (5) De gebruiker kan aan de interface niet zien wat de mogelijkheden van het programma zijn. (6) Afbeeldingen en kleuren worden niet of nauwelijks gebruikt. (7) De voorstelling die de gebruiker van de interface heeft, is vaak die van een intelligente computer, waarmee je "gewoon" kunt communiceren.

Voor de beginnende gebruiker is de passieve interface niet aantrekkelijk. De manier van opdrachten geven maakt echter veel goed. Net als bij een menselijke expert kunnen vragen op een natuurlijke manier worden

gesteld en zullen de antwoorden begrijpelijk zijn. Een voorbeeld hiervan is SAGE [Roth, 91], een systeem voor *automatic explanation*, dat veranderingen in kwantitatieve modellen probeert te verklaren. De gebruiker kan het programma vragen stellen in het Engels, die vervolgens ook in natuurlijke taal worden beantwoord, waar nodig voorzien van grafische voorstellingen. Een natuurlijke taal-interface is voor deze toepassing het meest voor de hand liggend. Maar ondanks het gemak waarmee een natuurlijke taal-interface bediend kan worden, hebben beginners er bepaalde moeilijkheden mee. Ten eerste weet de beginnende gebruiker van een systeem vaak niet hoe een vraag die hij in gedachten heeft geformuleerd moet worden om de gewenste informatie op het scherm te krijgen. Ten tweede is er de vraag welke woorden gebruikt kunnen worden, dus welke woorden het systeem kent. Ten derde heerst er vaak verwarring over de logische operatoren AND en OR. Veel mensen verwarren de logische "en" met de "en" uit de natuurlijke taal. Ten vierde formuleert de gebruiker zijn zinnen in het begin vaak zo simpel mogelijk, ook al kan het systeem met vrij complexe structuren overweg. Dit wordt echter pas later ontdekt. En ten vijfde bevatten de vragen vaak typefouten of is de opbouw onjuist. Dit kan te wijten zijn aan het feit dat de vragen tijdens het intikken worden geconstrueerd [McGraw, 92]. Afhankelijk van het correctie-mechanisme van de interface kunnen deze fouten automatisch worden verbeterd of moet de gebruiker expliciet aangeven wat hij bedoelt. Natuurlijke taal-interfaces zijn visueel meestal niet aantrekkelijk en verraden niet wat hun mogelijkheden zijn. De gebruiker zal zich bij de interface het beeld van een geautomatiseerde expert voorstellen. Dat helpt hem echter niet om de bediening te vereenvoudigen. Kortom: voor beginnende gebruikers is een natuurlijke taal-interface niet zo'n goede keuze; de bediening is vrij lastig en het uiterlijk is niet uitnodigend.

8.3.3. Menu-georiënteerde interface

Het werken met menu's bestaat voornamelijk uit het maken van selecties. Dit gebeurt met een aantal toetsen op het toetsenbord, met de muis of met een ander, soortgelijk aanwijsapparaat. Bij de menu-interface kunnen verschillende soorten menu's voorkomen, waarvan de belangrijkste zijn [McGraw, 92]:

- Het permanente menu: dit is altijd in beeld en bevat de commando's van het hoogste niveau.
- Het pull-down menu: dit is een menu dat meestal bij een commando uit het permanente menu hoort en op het scherm verschijnt alsof het een rolgordijn is dat naar beneden wordt getrokken.
- Het pop-up menu: dit menu verschijnt meestal in het midden van het scherm met een vraag aan de gebruiker.
- Het step-down menu: dit menu is gebaseerd op een boomstructuur; het selecteren van een bepaalde menukeuze kan een tweede step-down menu activeren (naast het eerste) en deze kan een derde activeren, enzovoort.
- Het multiple-choice menu: dit is een menu met bijvoorbeeld namen, situaties of omschrijvingen waaruit de gebruiker een keuze moet maken. Het multiple-choice menu was het eerste menu dat in programma's verscheen. Het kiezen van een optie in het menu gebeurde in het begin door een nummer of een letter in te tikken. Nu kan dat tevens met de muis of het toetsenbord.

(1) Bij een user interface met multiple-choice menu's heeft het programma de leiding. De gebruiker wordt met behulp van vragen stap voor stap door het programma geleid. Bij software met andere menu-soorten is het de gebruiker die beslist wat de volgende stap is. (2) Een opdracht wordt gegeven door een menu te activeren en daaruit een keuze te maken. (3) In een menu-gestuurd systeem wordt de voortgang van processen meestal in een "message box" weergegeven. Dit is een window dat tijdens het uitvoeren van een proces zichtbaar is en meldt welk deel (percentage) van het proces voltooid is. Foutmeldingen worden meestal ook in windows weergegeven, vaak met de mogelijkheid om de fout te herstellen of de opdracht nogmaals uit te voeren. De terugkoppeling is dus in het algemeen goed. (4) De interface is redelijk tolerant: ongewenste opdrachten kunnen vaak worden afgebroken en de kans op typefouten is klein door het gebruik van menu's. Bij dit soort interfaces bestaat slechts de kans dat de gebruiker een verkeerde keuze maakt. De menu-keuzes moeten dus zó worden geordend, dat fouten niet fataal zijn. Daarom mogen de veel gebruikte keuzes niet in de buurt van destructieve keuzes worden geplaatst en moet het programma bij potentieel gevaarlijke opdrachten om bevestiging vragen. Dat laatste wordt vrijwel altijd gedaan. (5) In een menu-gestuurd programma is het voor de gebruiker duidelijk welke opdrachten er gegeven kunnen worden. Alle mogelijkheden bevinden zich namelijk in één of meer menu's en de menu-keuzes die in een bepaalde situatie niet van toepassing zijn kunnen niet worden geselecteerd. De menu's zijn dus een geheugensteun voor de gebruiker. (6) Omdat er bij een menu-georiënteerde interface meer op het scherm staat dan bij een tekstinterface, worden de zichtbare elementen van het programma meestal in verschillende kleuren weergegeven. Dit verbetert de overzichtelijkheid.

Afbeeldingen komen in menu-gestuurde interfaces bijna niet voor. (7) Keuzemenu's kunnen aan menu-kaarten van restaurants worden gerelateerd, maar deze vergelijking gaat snel mank. Van een echte metafoor is geen sprake.

Veel expertsystemen hebben een menu-interface. Als de doelgroep van een expertsysteem niet uit de informaticawereld komt, is dit een efficiënte oplossing, omdat de gebruiker de interface (en dus het programma) stap voor stap kan leren kennen. Bij heel veel expertsystemen wordt gebruik gemaakt van multiple-choice menu's of menu's met vragen. De reactie van de gebruiker op het huidige menu is dan bepalend voor het volgende menu. Zo verkrijgt het systeem de informatie die het nodig heeft en wordt de gebruiker door het programma heen geleid. Deze techniek is zeer populair bij bijvoorbeeld medische diagnosesystemen zoals MYCIN (zie § 5.2.1.). Maar er zijn ook veel expertsystemen waarbij het programma een passieve rol heeft. Een goede menu-gestuurde interface laat zien welke mogelijkheden de gebruiker op ieder moment heeft en geeft bij het selecteren een omschrijving van de betreffende opdracht. Dit is voor de beginner een prettige vorm van assistentie. Bovendien is er op ieder moment slechts een beperkt aantal opdrachten beschikbaar, hetgeen helpt overzicht over het systeem te houden. Als het lijstje mogelijkheden niet de opdracht bevat die de gebruiker zoekt, moet er al eerder een andere keuze worden gemaakt. Zelfs de beginner die geen enkele opdracht kent, kan door het zoeken in de menu's een bepaalde taak toch laten uitvoeren. Deze interface-vorm geeft een goed beeld van wat de gebruiker aan het doen is en zorgt dat er weinig fouten kunnen optreden. De menu-structuur maakt op de gebruiker een opgeruimde, overzichtelijke indruk, die nog versterkt wordt als de ontwerper tactvol van kleuren gebruik heeft gemaakt. Ook al kan de gebruiker de menu-interface niet met iets uit zijn dagelijkse omgeving vergelijken, toch kan hij snel met de menu's leren werken. Er is dus sprake van een prettig uitziende interface en een eenvoudige bediening. Voor beginners is dit type user interface een goede keuze.

8.3.4. Grafische user-interface

Het laatste type interface dat hier wordt besproken, is de grafische user-interface (GUI). De grafische user-interface bezit de volgende eigenschappen: de wereld wordt gerepresenteerd door objecten en acties, de muis wordt gebruikt in plaats van het toetsenbord en de acties zijn direct, omkeerbaar en geven onmiddellijke feedback. Dit type interface valt daarom onder de zogenaamde *direct manipulation interfaces*. De gebruiker werkt direct met documenten en programma's in de vorm van iconen, menu's en windows. GUI's zijn populair als interface bij operating systems (bijvoorbeeld MacOS, X Windows, DECwindows en Windows NT) en andere programmatuur, zoals expertsystemen [McGraw, 92].

Hier volgen de antwoorden op de zeven vragen. (1) Bij GUI's is het de gebruiker die bepaalt wat er gebeurt. (2) Hij geeft opdrachten door met de muis iconen, buttons en menu-keuzes "aan te klikken". (3) Net als bij de menu-interface geven windows meestal weer wat het systeem aan het doen is. Een prettige conventie bij grafische, muisbestuurde systemen is het veranderen van de muiscursor in een klokje of een zandloper als het programma bezig is en even geen opdrachten kan ontvangen. Hierdoor weet de gebruiker dat het geen zin heeft om met de muis op objecten te "klikken". In combinatie met *message windows* (en vaak ook geluiden) is dit de beste vorm van feedback. (4) De meeste GUI's geven de mogelijkheid tot het afbreken van opdrachten. De kans op fouten is vrij klein, vanwege de "direct manipulation". De gebruiker verplaatst, kopieert of verwijdert objecten op het scherm door ze op te pakken met de muis en ergens anders te laten vallen (waardoor de bestanden die bij de objecten horen ook daadwerkelijk gemanipuleerd worden). Daardoor heeft hij een duidelijk beeld van zijn acties en maakt hij minder fouten. De interface vraagt bij destructieve acties om bevestiging. (5) Nog meer dan bij een menu-systeem geldt dat de gebruiker kan zien welke mogelijkheden er zijn. Alle buttons, menu's, *check boxes* en *radio buttons* impliceren bepaalde keuzes. Daardoor krijgt hij in één oogopslag te zien wat er geantwoord, ingesteld of gekozen kan worden. Tenminste: als het ontwerp van de interface goed is, zodat alle elementen op de juiste wijze worden benut. Er zijn geen opdrachten of functies die moeilijk te onthouden zijn of moeilijk te vinden. Wat de gebruiker ziet, kan hij gebruiken; niet meer en niet minder. (6) De grafische elementen van een GUI zijn vaak kleurige, herkenbare plaatjes, die uitnodigen om ermee te werken. De aantrekkingskracht komt waarschijnlijk voort uit de gewoonte van mensen om visuele mentale voorstellingen te creëren van objecten en concepten. Ze kunnen zich daardoor makkelijk met de interface vereenzelvigen. (7) Verreweg de meeste GUI's maken gebruik van de desktop-metafoor, de vergelijking met een bureaublad. De bestanden worden daarbij op het scherm voorgesteld als documenten in dossiermappen, die ook op een bureau kunnen liggen. Verder kunnen er op het scherm (bureau) een rekenmachine, een notitieblok,

een agenda en een printer aanwezig zijn [11]. Voor beginners blijkt zo'n metafoor het werken met de interface een stuk eenvoudiger te maken. De handelingen binnen het programma hebben een duidelijke relatie met de handelingen die achter een bureau worden verricht.

Programma's die zijn voorzien van een GUI zijn meestal makkelijk aan te leren. De gebruiker hoeft niet veel functies te onthouden omdat hij kan zien wat er mogelijk is en hij onthoudt de handelingen omdat hij ze begrijpt, omdat ze lijken op "echte" handelingen uit het dagelijks leven. Met de muis opdrachten geven is simpel en toch kunnen er complexe taken mee worden uitgevoerd. Het samenstellen van vragen (queries) wordt bij sommige systemen gedaan door de juiste iconen achter elkaar te plaatsen en met elkaar te verbinden. GUI's hebben over het algemeen een goede feedback naar de gebruiker en zijn vrij tolerant, want doordat de gebruiker zich bij de acties met de muis iets kan voorstellen, is de kans op een verkeerde opdracht klein. In ieder geval kleiner dan bij het geven van commando's. En de invloed van typefouten is bij het werken met de muis nihil. Een nadeel voor de beginnende gebruiker kan zijn dat hij zelf de leiding heeft. Als het scherm is gevuld met grafische voorstellingen, bestaat de kans dat hij in de war raakt en niet goed meer weet op welk element hij zich moet concentreren. Een ander nadeel is dat plaatjes niet universeel gebruikt kunnen worden om zaken weer te geven. Afbeeldingen hebben door cultuur– of taalverschillen niet voor iedereen dezelfde waarde en herkenbaarheid. Iconen zijn vaak gebaseerd op benaming en en dus taalafhankelijk. Een voorbeeld hiervan is een afbeelding van een raket die aangeeft dat een programma kan worden gestart. In het Engels is de opdracht "launch the word processor" vrij normaal, maar in het Nederlands heeft niemand het over het lanceren van de tekstverwerker. Maar deze nadelen wegen niet op tegen de voordelen. Een GUI is een uitstekend front-end voor bijvoorbeeld diagnostische systemen. Het kan de gebruiker een menu-structuur bieden met daarbij een weergave van het probleemgebied, zoals bijvoorbeeld een plattegrond van een fabriek met daarop de machines, zodat de gebruiker middels deze weergave instellingen voor de machines kan veranderen door ze te selecteren met de muis. In een grafische user interface worden de voordelen van een menu-gestuurd systeem gecombineerd met een grafische weergave, muisbesturing, meer visuele elementen en meer flexibiliteit. Dit type interface biedt dus een visueel aantrekkelijk uiterlijk en een eenvoudige bediening en is geschikt voor gebruikers met weinig computerkennis.

8.3.5. Combinaties van interfaces

Meestal is het voordelig om verschillende interface-vormen met elkaar te combineren. Als gebruikers een systeem goed kennen, ervaren ze behulpzaamheid van het systeem vaak als hinderlijk. De gevorderde gebruiker van een menu-gestuurd systeem of een GUI kan bepaalde menu's bijvoorbeeld als een beperking zien en zal (overbodige) hints die getoond worden eerder irritant dan behulpzaam noemen. Voor hem is het prettig als het systeem de mogelijkheid biedt menu's over te slaan door bijvoorbeeld bepaalde toetscombinaties te gebruiken. Zo kunnen opdrachten gegeven worden, zonder dat de gebruikers verplicht zijn een serie selecties te maken. Vrijwel alle GUI's hebben naast de muisbesturing zogenaamde "short cuts"; toetscombinaties die het equivalent zijn van één of meer menu-keuzes. Meestal kan de gebruiker zelfs overschakelen op een volledig commando-gestuurde interface, die hem de grootst mogelijke vrijheid biedt.

Een ander voorbeeld van een gecombineerde interface is het door Texas Instruments ontwikkelde NLMenu [McGraw, 92]. Dit programma kan worden gebruikt als een natuurlijke taal-interface voor kennissystemen. Het laat de gebruiker vragen in natuurlijke taal samenstellen door gebruik te maken van menu's met woorden en zinsdelen. Deze combinatie lijkt de voordelen van beide soorten interfaces te verenigen tot een makkelijk bruikbaar front-end, waarmee zowel de beginner als de gevorderde uit de voeten kan.

8.4. Keuze van de interface voor SIX

Op grond van de vergelijking van user interfaces in paragraaf 8.3. kan een keuze voor de interface van SIX worden gemaakt. Uitgaande van de zeven beoordelingsvragen met betrekking tot de bediening en het uiterlijk van de interface, vervallen de commando-interface en de natuurlijke taal-interface. Het meest geschikt voor SIX lijkt een menu-georiënteerde of een grafische user interface. Aangezien SIX wordt geïmplementeerd in de grafische shell G2, valt de keuze op de GUI. De vorm van de interface staat beschreven in paragraaf 10.3. (in

[11] Vreemd genoeg bevindt zich op het "bureau" ook vaak een prullenbak.

hoofdstuk 10 wordt het ontwerp van SIX besproken). De hardware die bij de bediening van de interface kan worden gebruikt, bestaat uit een grafisch kleurenscherm, een toetsenbord en een muis.

Er zal voor dit project geen combinatie van user interfaces worden gebruikt. Een combinatie biedt meestal wel voordelen, maar omdat SIX een prototype wordt en omdat de beschikbare tijd beperkt is, zal de interface van SIX volledig grafisch georiënteerd zijn. Het ontwikkelen van een geheel nieuw type interface valt uiteraard ook buiten het bestek van dit project. Bovendien is dat niet nodig want de bestaande GUI's bieden voldoende mogelijkheden voor een front-end voor SIX.

8.5. Evaluatie

Als we de zeven vragen over de bediening en het uiterlijk nu op de interface van SIX toepassen, is dit het resultaat: (1) De gebruiker heeft de leiding. Er is dus wel enige kennis van het pakket nodig, maar deze hoeveelheid is niet noemenswaardig. (2) Opdrachten worden gegeven met de muis. Gegevens moeten worden ingevoerd met het toetsenbord. (3) Afgezien van de problemen die in de vorige alinea zijn besproken, geeft het systeem in de meeste situaties aan wat er van de gebruiker wordt verwacht. (4) Typefouten in de gegevens (zoals een letter in een getal) worden stilzwijgend verwijderd en andere fouten worden gemeld. (5) De gebruiker ziet altijd wat hij kan doen, omdat er in het hoofd-window en in de dialoog-windows hints worden gegeven. Het invoeren van de gegevens (in het hoofd-window) gebeurt met een schema van een papiermachine en is overzichtelijk (mits de gebruiker beschikt over de basiskennis van papiermachines, die voor de doelgroep mag worden verondersteld). (6) Het schema van de papiermachine is grafisch en kleurig. Kleuren worden verder sporadisch, maar consequent toegepast. (7) De gebruiker kan zich bij de interface een papiermachine voorstellen, waarvoor op een aantal punten gegevens moeten worden ingevoerd. Deze punten in het schema van de machine zijn de meetpunten bij de werkelijke papiermachine.

Door het gebruik van een (herkenbaar) papiermachineschema, dat bestaat uit gekleurde afbeeldingen biedt SIX een visueel aantrekkelijke user interface. Omdat het invoeren van gegevens makkelijk is, evenals het laten analyseren ervan, biedt SIX een eenvoudige user interface. Dit waren de gestelde voorwaarden voor een toegankelijk voorbeeldsysteem, dat ook door gebruikers met weinig computerkennis moest kunnen worden bediend. En met uitzondering van een feedback-probleem komt de user interface van SIX aan deze voorwaarden tegemoet.

Voor het maken van user interfaces in G2 levert Gensym de grafische ontwikkelhulp GUIDE. Deze maakt gebruik van UIL, de User Interface Library (ook van Gensym). GUIDE en UIL werden gebruikt om de interface van SIX te implementeren. Met deze hulpmiddelen kan op redelijk eenvoudige wijze een interface worden gemaakt. De snelheidswinst tijdens het programmeren viel (in vergelijking met andere GUI-hulpmiddelen) echter tegen. Ook bij het gebruik van het ontwikkelde produkt is er een nadeel op het gebied van snelheid: het opbouwen van de dialogen voor boodschappen aan en interactie met de gebruiker duurt vele seconden, waardoor er een ernstig feedback-probleem ontstaat. De maximaal toelaatbare responstijd voor het oproepen van een dialoog is namelijk 2 seconden [Willemse, 88]. En de gebruiker weet niet dat het systeem met het maken van een dialoog-window bezig is, omdat G2 de conventie van de "zandloper-cursor" niet gebruikt. De langzame dialogen vormen een probleem in de huidige versie van SIX. De andere grafische elementen die UIL biedt, zoals invoervelden en scroll area's, zijn ook niet prettig in het gebruik. Het hele grafische pakket is aan een verjonging toe. Maar aangezien G2 nog steeds wordt verbeterd, zal UIL ongetwijfeld ook nog verder worden verbeterd.

9. GENSYM G2

R.A. Carton: § 9.1., § 9.2.; M. Pruis: § 9.3.

Het bedrijf Gensym Corporation dat zijn oorsprong in de Verenigde Staten heeft, brengt het pakket G2 op de markt. G2 is een uitgebreid gereedschap voor onder andere simulatie en geautomatiseerd redeneren. In dit hoofdstuk wordt een algemene beschrijving van de mogelijkheden van het pakket gegeven, gevolgd door een uiteenzetting van de mogelijkheden en beperkingen van de *objectgeoriënteerdheid* van G2. De laatste paragraaf behandelt het deel van G2 dat gericht is op de ontwikkeling van expertsystemen. Als referentie voor de syntaxis is in bijlage D de language reference voor G2 opgenomen. Voor meer informatie over G2 wordt de lezer verwezen naar [Gensym, 92]. Dit hoofdstuk geeft een indruk van de (on)mogelijkheden van G2 en dient als basis voor het hoofdstuk over implementatie.

9.1. Algemeen

G2 is een programma voor het ontwikkelen van real-time expertsystemen. De combinatie van redeneermogelijkheden en tijdsgebonden programmeren maakt G2 interessant voor het besturen, controleren en simuleren van processen. Het pakket is objectgeoriënteerd en heeft een grafische interface met muisbesturing en windows. G2 is een *interpreter*. Daarom is het niet mogelijk om zelfontwikkelde programmatuur los van G2 te gebruiken. Ook heeft een interpreter het nadeel dat de snelheid van de programma's niet zo hoog is als bij een compiler. Bovendien wordt (bij deze interpreter) alle programmacode onderdeel van het systeem. Programmeren is niet het schrijven van een opzichzelfstaand programma, maar het uitbreiden van het systeem.

Het programmeren gebeurt in een taal die op Pascal lijkt, maar een stuk uitgebreider is. G2 kent commando's zoals if-then-else, while, case en for, maar het for-commando zou bijvoorbeeld deze vorm kunnen hebben:

for ad = each additive do ...

Dit is een makkelijke manier om bewerkingen op alle "additives" (additieven) te laten uitvoeren. Net als in Pascal zijn er functies en procedures, die overigens kunnen worden verlaten met return. De programmeur kan zijn eigen foutafhandelingsprocedures installeren met het statement on error. In statements kunnen ook tijd-gerelateerde uitdrukkingen voorkomen, zoals bij wait:

wait until x = 0 checking every 5 seconds

Of bij het statement do in parallel:

```
do in parallel until one completes
    statement1;
    statement2;
    ...
end
```

En er zijn negen expressies die met waarden van variablelen gedurende een bepaald tijdinterval overweg kunnen, zoals bijvoorbeeld:

the average value of x during the last 10 minutes

Hieruit blijkt ook dat G2 breedsprakig is. De constructies variëren van eenvoudige toekenningen zoals dat in C gebruikelijk is, tot uitgebreide opdrachten die soms meer op commentaar bij de code lijken dan op de code zelf. Een voorbeeld hiervan is:

if there exists a workspace W such that (the name of W = information) then
show W at the center of the screen

In dit voorbeeld wordt over een workspace gesproken. G2 maakt hiervan gebruik om code en gegevens te structureren en in blokken aan te kunnen spreken. Een workspace is een window met daarop procedures, regels, variabelen, ikonen of andere G2-objecten. Workspaces limiteren het bereik van code en gegevens niet, maar zorgen voor overzicht en maken het aanspreken van groepen gegevens makkelijker. In SIX komt bijvoorbeeld de volgende opdracht voor:

for st = each starch upon current-project-data do ...

"Current-project-data" is de naam van de workspace waarop de door de gebruiker ingevoerde gegevens komen te staan. Met de referentie "upon" kunnen de starches (zetmelen) worden gemanipuleerd. Als het "upon"-deel wordt weggelaten, zal het systeem niet slechts naar de ingevoerde zetmelen kijken, maar ook naar alle zetmeeldefinities die tot de standaardkennis van SIX behoren, omdat dit ook starches zijn. Workspaces en andere objecten kunnen ook subworkspaces hebben. Daarop kunnen bijvoorbeeld regels worden gezet die op bepaalde momenten moeten worden ge(de)activeerd (= (niet) worden gebruikt tijdens een inferentieproces) met:

(de)activate the subworkspace of x

Als een applicatie gereed is, zal de programmeur ervoor zorgen dat de gebruiker alleen de workspaces te zien krijgt die voor hem van belang zijn. De workspaces met definities, code en andere zaken die alleen bij de ontwikkeling worden gebruikt, worden ontoegankelijk gemaakt.

Per workspace kunnen restricties worden ingesteld. Die geven aan hoe de objecten op een workspace reageren op muiscommando's. Het gebruik van restricties heeft alleen zin in combinatie met het gebruik van "modes" in G2. Het programma staat standaard in de zogenaamde administrator mode. Als deze aktief is, kan de gebruiker alle instellingen wijzigen en dus ook de restricties. Een mode is niet veel meer dan een naam waaraan wordt gerefereerd in de restricties. Nieuwe modes kunnen daarom makkelijk worden gecreëerd: het opnemen van de naam in een restrictie is voldoende. De restricties van een workspace kunnen er bijvoorbeeld zó uitzien:

unless in administrator mode:
non-menu choices for papermachine-part exclude: move-objec t;
when in end-user mode:
menu choices for kb-workspace exclude: delete-workspace

In dit voorbeeld staat dat het verplaatsen van het object "papermachine-part" niet tot de mogelijke bewerkingen ("non-menu choices") op het object behoort, tenzij G2 in administrator mode is geschakeld. Verder wordt aangegeven dat er een mode "end-user" bestaat, waarin de menukeuze "delete workspace" niet zichtbaar mag zijn in het menu van de kb-workspace (waarbij "kb" voor knowledge base staat).

Bij het invoeren van programmacode geeft G2 aan welke woorden op iedere plaats kunnen worden gebruikt. Als in een procedure bijvoorbeeld "if" wordt getypt, verschijnt er een lijst met de woorden die erachter kunnen staan, zoals "there exists", "the class of" en "the array-length of". Als de programmeur één van deze mogelijkheden aanklikt, wordt de tekst automatisch in het edit-window geplaatst. Na "if" kan ook een variabele worden gebruikt en daarom zal in de lijst ook "any variable" staan. Als hierop wordt geklikt, verschijnt een lijst met alle variabelen die op dat moment in gebruik zijn, waaruit gekozen kan worden. Natuurlijk is het ook mogelijk een nieuwe variabele in te typen. Als er echter bij een woord uit de taal van de mogelijkheden wordt afgeweken, zal G2 dit meteen melden en is het niet mogelijk om de procedure op te slaan voordat de fout hersteld is.

9.2. Object-oriëntatie

Het objectgeoriënteerde karakter van G2 maakt onder andere overerving (inheritance) mogelijk. Objecten kunnen van andere objecten worden afgeleid en hebben daardoor gemeenschappelijke attributen. Voor bijvoorbeeld workspaces is overerving van restricties handig: als workspace X van Y is afgeleid, heeft X dezelfde restricties als Y en eventueel nog andere. Zo kan bovenin de workspace-hiërarchie worden aangegeven hoe alle workspaces zich gedragen. Dit geldt ook voor andere objecten: omdat bovenin de objecthiërarchie van G2 het "item" staat (hiervan is onder andere "object" afgeleid) en omdat deze bijvoorbeeld het attribuut "name" heeft, weet de programmeur dat alle objecten dit attribuut hebben.

Objecten kunnen wel attributen hebben, maar geen *methodes*. Dit is een tekortkoming van versie 3.0 van G2. Het is niet mogelijk procedures en functies bij een klasse van objecten te laten horen. Er is in deze versie dus ook geen sprake van polymorfisme. Eén van de voordelen van objectoriëntatie is het onderbrengen van programmacode in objecten, zodat het niet nodig is verschillende procedures te schrijven die bijna hetzelfde doen, maar voor verschillende objecten. Bij een goed objectgeoriënteerd systeem kan er bijvoorbeeld één methode "write" zijn, waaraan een object wordt meegegeven. Als dit object zetmeel (starch) is, wordt de code van "write" voor zetmeel aangeroepen en als het een additief (additive) is, wordt de "write" van additief geactiveerd. Dit heet polymorfisme. In conventionele systemen echter (en ook in G2) zullen de procedures "write-starch" en "write-additive" nodig zijn. Een wijziging in "write" van het ouderobject van zetmeel en additief heeft effect in de beide kinderobjecten, terwijl voor dezelfde verandering in een G2-applicatie allebei de write-procedures moeten worden aangepast. De benaming "object oriented" is daarom voor G2 versie 3.0 niet passend. De nieuwe versie (4.0) van de ontwikkelomgeving zal wel polymorfisme ondersteunen en zelfs multiple inheritance, waarbij een object van meer dan één ouderobject kan worden afgeleid en dus ook de functionaliteit van meerdere objecten kan erven. Versie 4.0 bevindt zich nu in de testfase.

Er is nog een probleem waar we bij het maken van SIX tegenaan zijn gelopen. In het programma zou gebruik worden gemaakt van een object dat als attribuut een ander object heeft, namelijk een additive (additief). Het bleek niet mogelijk te zijn om aan dit attribuut een starch (zetmeel) toe te kennen in plaats van een additive. Maar starch is direct afgeleid van additive en zou zich dus hetzelfde moeten gedragen. Een "echt" objectgeoriënteerd systeem[12] zou deze toekenning wel ondersteunen. We moesten een noodoplossing bedenken, namelijk: extra objecten creëren.

9.3. Expertsysteem-ontwikkelomgeving

Deze paragraaf beschrijft hoe het specifieke expertsysteemdeel van de ontwikkelomgeving werkt. Paragraaf 9.3.1. behandelt de opbouw van de produktieregels. Het tweede deel gaat dieper in op de technieken die G2 ondersteunt om te redeneren met de produktieregels.

9.3.1. Produktieregels

Door middel van regels kan in G2 gedefinieerd worden hoe een applicatie moet reageren op bepaalde condities [Gensym, 92]. Een regel bestaat uit een *antecedent* en een *consequent*. Het antecedent specificeert de condities waaraan een regel moet voldoen en de consequent specificeert welke acties er ondernomen moeten worden als aan de condities voldaan is. Als G2 een regel aanroept dan wordt eerst de logische expressie in het conditionele deel van het antecedent geëvalueerd. Als dit de waarde "true" oplevert, worden de acties uit de consequent uitgevoerd. In de consequent van een regel kunnen bepaalde acties voorkomen. De belangrijkste zijn hieronder vermeld:

[12] Ook al heeft een systeem nog zoveel objectgeoriënteerde eigenschappen, "echt" objectgeoriënteerd is het niet. Omdat de taal waarin het is geschreven of de computer waarop het draait niet objectgeoriënteerd is, moeten er altijd concessies worden gedaan. Zo zal bijvoorbeeld het object dat zich bovenin de hiërarchie bevindt niet te wijzigen zijn, omdat het geen object is, maar er alleen uitziet als een object.

- *Conclude*: Deze actie verandert de waarde van array's, lijsten, attributen, variabelen, parameters en relaties. Veel regels in een expertsysteem zullen de conclude actie in de consequent hebben. Hiermee kan naar aanleiding van een gegeven (antecedent) een andere waarde aangepast worden.
- *Inform*: Stuurt tekst naar een bepaald window (workspace/messageboard).
- *Insert/Remove*: Toevoegen en verwijderen van elementen uit een lijst.
- *Invoke*: Activeert een bepaalde categorie regels voor een bepaald object.
- *Start*: Activeert een procedure.

Een aantal van deze acties wordt nog uitgebreider besproken in de volgende paragraaf. Voor een complete lijst van acties die in een consequent opgenomen kunnen worden wordt de lezer verwezen naar bijlage D die de language reference G2 bevat [Gensym, 92]. Voorbeelden van mogelijke regels in een G2-systeem zijn:

if the level of tank-a < 5 then conclude that tank-a is empty and inform the operator that "Tank-A is empty."

when sensor-5 is broken then inform the operator that "[the name of sensor-5] is broken. I'm taking corrective action." and invoke repair rules for sensor-5

In de voorbeelden hierboven zijn al twee soorten regels afgebeeld. Binnen G2 bestaan 5 soorten regels, nl.:

- *if* regels
- *initially* regels
- *unconditionally* regels
- *when* regels
- *whenever* regels

Van alle regelsoorten kan door het gebruik van een *for any/if any* constructie voor een regel een generieke variant gemaakt worden. Generieke regels zijn regels die betrekking hebben op meer dan één object uit de applicatie (meestal een groep of klasse objecten). De *for any/if any* constructie geeft aan op welke set objecten de regel van toepassing is. Een voorbeeld van een generieke regel is [Gensym, 92]:

for any heat-exchanger
if the inlet-temp of the heat-exchanger < 50 F
then focus on the heat-exchanger

Generieke regels kunnen twee verschillende vormen hebben. Er kunnen *if any* en *for any* regels opgenomen worden in een knowledge base. In het eerste geval zal het systeem kijken of er een object (van de genoemde klasse/groep) aanwezig is waarop de regel kan worden toegepast en de regel daarop toepassen. In het tweede geval zal op alle objecten die in de genoemde groep vallen en aan de condities voldoen de regel worden toegepast. Generieke regels kunnen ook in geneste vorm voorkomen:

for any tank
for any valve connected to the tank
if the valve is broken
then inform the operator that "[the name of the valve] is broken"

Als deze generieke regel wordt uitgevoerd kijkt G2 voor elke tank naar alle kleppen. De kleppen die gebroken zijn worden gemeld aan de operator (de acties in de consequent worden uitgevoerd voor dat tank/klep-paar) [Gensym, 92].

9.3.2. Het inferentie-mechanisme

De G2 (real-time) inference engine is het mechanisme dat het redeneren met de in de vorige paragraaf beschreven regels verzorgt. Het redeneert steeds over de huidige status van de applicatie en communiceert met de gebruiker of start andere activiteiten op basis van wat wordt afgeleid. De G2 inference engine opereert op basis van de volgende informatiebronnen:

1. De kennis uit de knowledge-base

2. Gesimuleerde waarden
3. Waarden ontvangen van externe bronnen (meetapparatuur etc.)

Kort samengevat kan de inference engine de volgende taken afhandelen [Gensym, 92]:

- Regels scannen op per regel ingestelde intervallen (*scanning*).
- Het "focussen" op sleutelobjecten, of klassen van objecten door bepaalde geassocieerde regels te starten (*focusing*).
- Bepaalde categorieën van regels starten voor een bepaalde klasse van objecten (*invoking*).
- Regels starten op basis van een gebeurtenis.
- Backward-chaining verzorgen voor het verkrijgen van waarden.
- Forward-chaining verzorgen als er waarden bekend worden.
- Reageren op gebeurtenissen van de gebruiker, simulator, of externe data-servers.
- Sequentiële af te handelen procedures starten.

Forward en backward-chaining zijn standaardfaciliteiten voor de meeste expertsysteem-shells waarvan de onderliggende theorie al is beschreven in hoofdstuk 7. De *scanning, focusing* en *invoking* functionaliteit zijn extra, voor real-time expertsystemen essentiële technieken. In deze paragraaf zal kort aangeven hoe G2 met deze technieken omgaat. Daar waar nodig zal de lezer verwezen worden naar de onderliggende theorie in hoofdstuk 7. Het "focussen", "invoken" van regels en prioriteiten toekennen aan regels valt onder de in dat hoofdstuk beschreven techniek met behulp van control-knowledge.

Het "scannen" van regels

De inference engine van G2 kan herhaaldelijk een regel aanroepen na regelmatige tijdsintervallen. Deze techniek wordt scanning genoemd. Het tijdsinterval tussen de aanroepen is het scan-interval van de regel. Elke regel in G2 bevat een aantal attributen waaronder een scan-interval attribuut. Hier kan de ontwikkelaar dus aangeven hoe vaak per tijdseenheid een regel aangeroepen moet worden. Deze techniek is erg nuttig voor zogenaamde intelligent control applicaties. Zo'n applicatie bekijkt continu een proces en onderneemt afhankelijk van het verloop in de metingenactie. Een gescande regel kan wanneer aan de conditie voldaan is de oorzaak zijn van het aanroepen van andere regels.

Wanneer een generieke regel een scan-interval toegewezen gekregen heeft wordt elke instantiatie van de regel één keer per tijdsinterval aangeroepen. Afhankelijk van het aantal objecten dat aangesproken wordt, de complexiteit van de regel en de instelling van het scan-interval kan het aantal regels dat aangeroepen wordt erg snel oplopen.

De "focusing" techniek

Wanneer de inference engine een focus-actie uitvoert op een bepaald object worden alle regels aangeroepen die het object opgenomen hebben in hun focal-object attribuut en alle regels die de klasse (of superklasse) van het object opgenomen hebben in hun focal-class attribuut. Wanneer een generieke regel een bepaald object als focal-object heeft en er wordt een focus actie uitgevoerd op dat object dan wordt de regel uitgevoerd voor alleen dat object. Een generieke regel werkt op dat moment als een specifieke regel.

Starten van categorieën regels (invoking)

Wanneer de inference engine een invoke actie uitvoert op een bepaalde categorie van regels dan worden alle regels die in die categorie vallen aangeroepen. De ontwikkelaar kan aangeven tot welke categorie een regel behoort in het category attribuut. Een voorbeeld is:

```
invoke safety rules
```

Hierdoor roept de inference engine alle regels aan die safety als waarde hebben voor het focal-category attribuut. De engine start deze regels voor alle objecten in de regel. De generieke regels kunnen tevens worden aangeroepen voor een bepaald object:

```
invoke safety rules for headbox
```

Wake-up van regels

Wanneer een variabele die als gevolg van een regel staat te wachten op een waarde die waarde ontvangt wordt door de inference engine de desbetreffende regel weer geactiveerd. Wanneer een regel bijvoorbeeld de waarde van een tijdsafhankelijke variabele nodig heeft en deze op dat moment geen geldige waarde heeft wordt er een verzoek naar de data-server gestuurd om een waarde te bepalen voor de variabele. Zolang de data-server geen uitsluitsel gegeven heeft doet de inference engine andere dingen. Wanneer de variabele een waarde krijgt, voert de engine een wake-up uit voor de regel.

Nadat een bepaalde tijd verstreken is zal de inference engine een regel niet meer opnieuw activeren. De betreffende tijdsduur kan ingesteld worden bij de regel (timeout-for-rule-completion) of bij de parameters van de inference engine (timeout-for-inference-completion).

Forward en backward chaining in G2

Het mechanisme van forward en backward chaining is al behandeld in hoofdstuk 7. G2 ondersteunt zowel forward als backward chaining. Voor bijna alle objecten binnen G2 kan ingesteld worden welke methoden gebruikt mogen worden voor het verkrijgen en/of door redeneren met die objecten. Twee manieren van backward chaining worden ondersteund: depth first en breadth first backward chaining . In het eerste geval probeert het systeem via steeds één mogelijke volgende regel zo veel mogelijk en diep mogelijk door te redeneren. De tweede manier bekijkt steeds voor elke regel alle mogelijke volgende regels alvorens van die regels weer alle mogelijke volgende regels te onderzoeken. Voor een regel kan worden aangegeven of deze aangeroepen mag worden als gevolg van forward danwel backward chaining . Whenever-regels kunnen standaard echter niet aangeroepen worden als gevolg van forward chaining . Ze kunnen dit wel veroorzaken.

Data seeking

Wanneer G2 een waarde voor een variabele nodig heeft, wordt eerste de laatst opgeslagen waarde onderzocht om te kijken of de ingestelde geldigheidsduur al verlopen is. Als de variabele niet verlopen (expired) is wordt de gevonden waarde gebruikt. Als de waarde wel verlopen is bekijkt G2 het data-server attribuut van de variabele. Als de variabele z'n waarde inderdaad van een data-server krijgt probeert G2 een nieuwe waarde van die server te krijgen (data seeking).

Alleen G2-variabelen veroorzaken data-seeking. Een parameter heeft in principe altijd een geldige waarde en kan dus geen data seeking veroorzaken. Een aantal gebeurtenissen kunnen de oorzaak zijn van data-seeking:

- Een variabele heeft een update-interval
- Een display object dat gebruik maakt van een variabele heeft een update-interval
- Een regel refereert aan een variabele
- Een declaratie, *collect data* commando of *wait until* statement refereert aan een variabele
- Een update-actie refereert aan een variabele

Wanneer een request voor een waarde niet gehonoreerd wordt door de data-server (de variabele krijgt geen waarde) heeft de variabele gefaald. Een variabele blijft geregistreerd staan als gefaald totdat er waarde toegekend wordt. Met behulp van whenever-regels kan geredeneerd worden op basis van het falen van variabelen.

Prioriteiten toekennen aan regels

Het door de G2 engine "scheduling" van een taak voor een regel kan beïnvloed worden door een prioriteit aan de regel toe te kennen. Dit is een vorm van control knowledge (zie hoofdstuk 7). Een priority-attribuut bepaalt wat de prioriteit is van de geplande taken in de scheduler. G2 roept de regel met de hoogste prioriteit het eerst aan. De regels met een lage prioriteit worden aangeroepen zodra de scheduler wat extra tijd beschikbaar heeft.

De inference engine ontvangt steeds een current-task-queue van de scheduler waarin de regels staan die in de volgende paar seconden, zodra er processortijd beschikbaar is, uitgevoerd worden. Deze regels zijn gesorteerd op prioriteit. Over het gereed komen van een regel uitvoering heeft dit geen invloed. Hierbij is meer de complexiteit (het aantal referenties naar variabelen) van een regel van belang.

10. ONTWERP SIX

R.A. Carton: § 10.1., § 10.3.; M. Pruis: § 10.2.

Voor het ontwerp van SIX waren drie zaken belangrijk. Ten eerste de kennisacquisitie , waarbij kennis werd verzameld die in het programma zou worden gebruikt. Ten tweede de manier waarop deze kennis is gestructureerd. En ten derde opzet van de user interface van SIX. Deze drie onderwerpen komen hier achtereenvolgens aan bod.

10.1. Kennisacquisitie

Voor het verzamelen van kennis hebben we experts op het gebied van de papierfabricage geïnterviewd. In totaal zijn er vijf interviews gehouden, waarvan de resultaten zijn opgenomen in bijlage B. Bij het samenstellen van de interviews is geen interviewtechniek gebruikt. De belangrijkste vragen werden opgeschreven en meestal kwamen er tijdens de gesprekken nieuwe vragen bij. Ieder gesprek nam ongeveer 1 tot 1½ uur in beslag en werd met Jan Hendriks en/of Jacob Terpstra van het Papierlab van AVEBE in Foxhol gehouden. De resultaten van de interviews werden bij Vertis uitgewerkt en ter goedkeuring aan de geïnterviewden voorgelegd. De uitwerkingen vormen onze basiskennis omtrent de fabricage van papier.

In de eerste interviews kwamen zaken naar voren die meer toelichting nodig hadden en daarom in latere interviews herhaald werden. Het aanpassen van de opdracht (zie hoofdstuk 2) is voortgekomen uit de eerste twee interviews, waarin bleek dat de wensen van de klant omtrent de eigenschappen van het geproduceerde papier niet zo uitgebreid zijn als werd aangenomen. In de laatste interviews werd vooral aandacht besteed aan het zoeken naar afwijkingen in een proces en het terugdringen ervan. Verder werd er gepraat over de meest voorkomende afwijkingen en de (vuist)regels voor het opsporen en aanpassen daarvan.

Naast de interviews zijn er ook gesprekken gevoerd met Jelmer Sjollema, de begeleider bij Vertis. De inhoud ervan had meestal betrekking op de chemische achtergrond van de papierfabricage. De belangrijkste onderwerpen waren de geleidbaarheid en de zetapotentiaal. Bovendien hebben we kennis uit een aantal boeken en verslagen gehaald, die bij Vertis en AVEBE beschikbaar waren (onder andere [Advokaat, 94], [Brouwer, 90] en [Smook, 92]).

10.2. Kennisstructuur (expertisemodel)

Zoals al eerder genoemd is in het begin van het project de Mini KADS methodiek gebruikt. Het drie-lagen model van Mini KADS vinden we dan ook terug in het kennismodel (of expertisemodel in Mini KADS) van SIX. De eerste laag is de domeinlaag. Deze laag bevat een beschrijving van de kennis die een rol speelt bij het oplossen van het probleem (gegevensbeschrijving). Aan de hand van interviews met experts van het AVEBE Papierlab (zie § 10.1.) is bekeken welke entiteiten van belang zijn voor het expertsysteem en wat de relatie is tussen die entiteiten. Deze laag bevat de definitie voor de gegevensverzameling (zie ook § 7.2.). De inferentielaag geeft inzicht in de structuur van de kennis. In deze laag wordt vastgelegd welke rol de in de domeinlaag vastgelegde kennis speelt in het redeneerproces. De structuur van de regels binnen het systeem wordt vastgelegd en objecten die van belang zijn voor het inferentieproces worden gedefinieerd. De derde laag, de taaklaag, gebruikt de structuur uit de tweede laag om tot een oplossing te komen voor verschillende soorten vragen aan het expertsysteem. In deze laag worden verschillende strategieën vastgelegd met behulp van

redeneerstappen uit de tweede laag. Deze laag is een uitbreiding op het model uit par agraaf 7.2. en wordt gebruikt om een systeem op een goede manier met meerdere probleemstellingen om te laten gaan. Afhankelijk van de taak van een systeem wordt een bepaalde strategie gevolgd bij het kiezen van de inferentiestappen.

In de opzet van het systeem valt de derde laag (de taaklaag) grotendeels weg, omdat het systeem maar één taak bevat (het optimaliseren van het proces). Deze taak kan wel onderverdeeld worden maar deze onderdelen worden in het huidige ontwerp niet voor andere taken gebruikt. De taaklaag wordt in dit hoofdstuk daarom ook niet verder apart besproken. Wel wordt op een aantal plaatsen in dit hoofdstuk aangegeven waar de functionaliteit van een taaklaag bij eventuele uitbreiding van het systeem terecht zou komen.

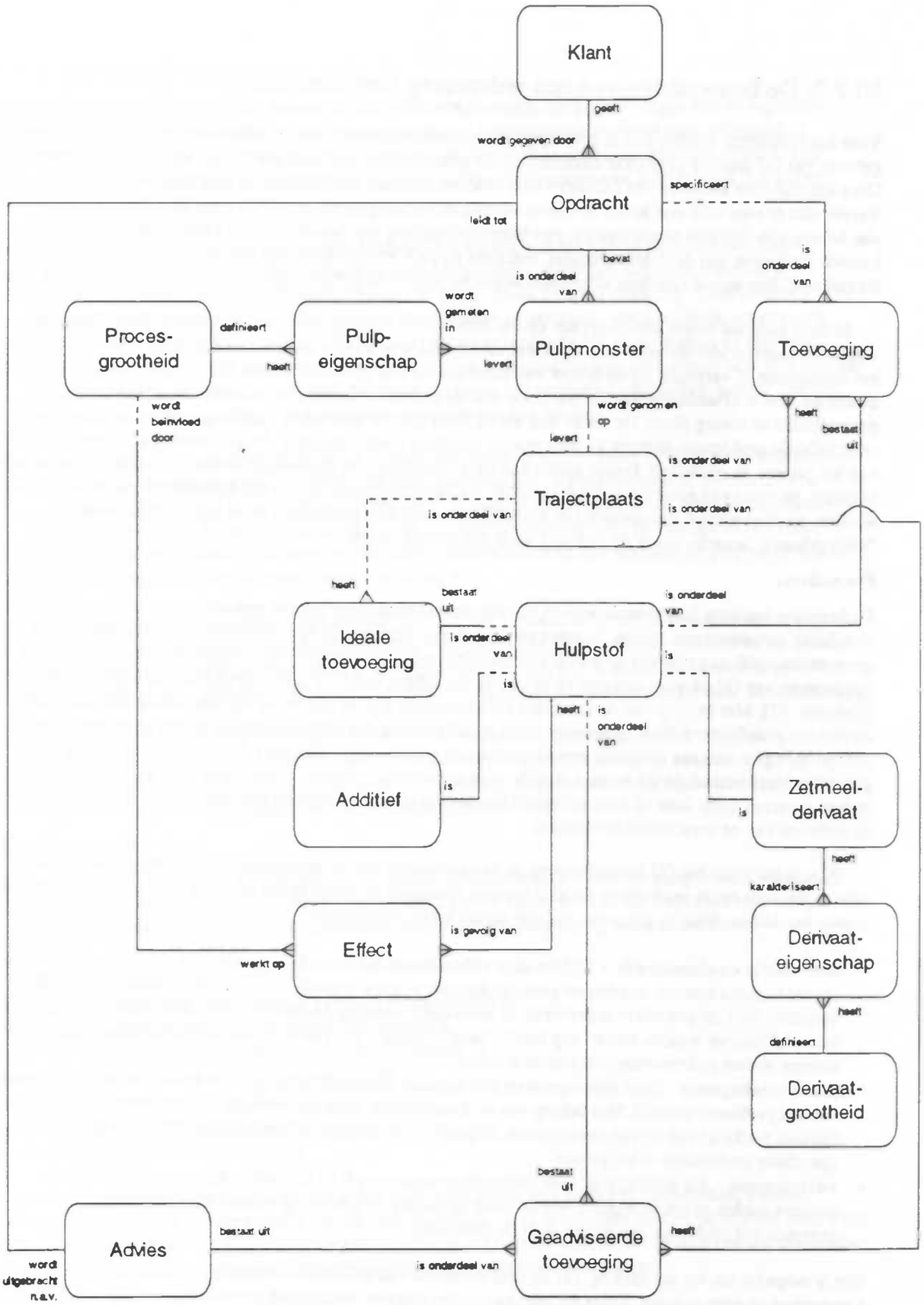
10.2.1. Entiteiten en relaties (domeinlaag of gegevensbeschrijving)

Door middel van een drietal interviews (paragraaf 10.1.) met experts van het Papierlab is vastgelegd welke entiteiten en relaties tussen deze entiteiten een rol spelen bij het optimaliseren van de papierfabricage (figuur 10.1.). Het E/R diagram waarin deze entiteiten en de relaties werden weergegeven is naar aanleiding van deze gesprekken een aantal malen herzien, mede omdat op een bepaald moment besloten is de papierwensen buiten beschouwing te laten en uit te gaan van proces-optimalisering. Om een goed beeld te krijgen van de entiteiten en relaties waarmee rekening gehouden moet worden in het systeem is met behulp van het E/R diagram een (genormaliseerd) relationeel datamodel opgesteld. Hieruit kan worden afgeleid welke entiteiten aanwezig zijn in de domeinlaag.

Klant	(klantnaam)
Opdracht	(opdrachtcode, klantnaam)
Pulpmonster	(monstercode, opdrachtcode, trajectplaatsnaam)
Trajectplaats	(trajectplaatsnaam)
Pulpeigenschap	(monstercode, procesgrootheid, waarde)
Toevoeging	(opdrachtcode, stofnaam, trajectplaatsnaam, waarde, eenheid)
Geadviseerde toevoeging	(adviescode, stofnaam, trajectplaatsnaam, waarde, eenheid)
Ideale toevoeging	(stofnaam, trajectplaatsnaam, waarde, eenheid)
Hulpstof	(stofnaam, categorie)
Zetmeelderivaat	(derivaat-stofnaam)
Additief	(additief-stofnaam)
Functie	(stofnaam, procesgrootheid, verandering)
Derivaateigenschap	(derivaat-stofnaam, derivaatgrootheid, waarde)
Procesgrootheid	(procesgrootheidnaam, eenheid)
Derivaatgrootheid	(derivaatgrootheidnaam, eenheid)
Advies	(adviescode, opdrachtcode)

Bij bovenstaande datamodel moet opgemerkt worden dat de sleutels "derivaat-stofnaam" en "additief-stofnaam" een waarde hebben uit een subset van de mogelijke waardes voor "stofnaam". Hiermee wordt onderscheid aangebracht tussen de additieven en een andere specifieke groep hulpstoffen, namelijk zetmeelderivaten waarvan meer eigenschappen bekend zijn.

Het hierboven beschreven relationele datamodel dient als gegevensbeschrijving van de knowledge base. Het model bevat alle objecten die nodig zijn om het papierfabricageproces en de gegevens die nodig zijn voor het geven van advies afdoende te beschrijven.



Figuur 10.1. Entiteit/relatie-diagram van het wet-end

10.2.2. De bouwstenen van een redenering (inferentielaag)

Voor het redeneren binnen SIX is geen apart formalisme ontworpen om de kennis in vast te leggen. Door het gebruik van G2 staat de gebruiker automatisch het hele krachtige kennisformalisme van G2 ter beschikking. Door het real-time karakter van G2 horen daar ook constructies bij die absoluut niet bruikbaar zijn binnen SIX. Verder zijn er voor SIX een aantal abstracte structuren ontworpen die specifiek voor de redenering binnen SIX van belang zijn. Dit zijn elementen die een kennistechnoloog met behulp van het G2-formalisme ook zelf had kunnen toevoegen aan de redeneerkennis, maar die zo toch zo algemeen zijn dat ze onderdeel zin van het formalisme. Een aantal van deze elementen wordt bijvoorbeeld gebruikt tijdens de algemene classificatiestap.

In deze pargraaf wordt kort ingegaan op het formalisme. Verder wordt een stappenverdeling besproken waarin de regels onderverdeeld worden. Deze onderverdeling in drie categorieën kennis is niet vastgelegd in het formalisme of verplicht bij de invoer van kennis maar een gereedschap voor het overzichtelijk opslaan van de kennis. Het is afhankelijk van de discipline van de kennistechnoloog in hoeverre deze onderverdeling gehandhaafd en zinnig blijft. De derde laag uit de Mini KADS methodiek (taaklaag) zou voor het oplossen van verschillende problemen gebruik kunnen maken van deze onderverdeling. In een, eventueel tijdens het vervolg van het project, te realiseren *kennis-editor* kan deze verdeling wel ingebouwd worden en moet tevens worden bekeken, op basis van ervaring met het systeem, welke beperkingen op het G2 formalisme ingebouwd moeten worden, en of er andere categorieën noodzakelijk om meerdere problemen op te kunnen lossen naast "optimalisatie" waardoor ook de taaklaag weer van belang wordt.

Formalisme

De kennistechnoloog kan gebruik maken van een aantal elementen om een redenering op te bouwen: regels, variabelen en parameters, acties, functies en procedures, kortom alle G2 elementen en de objecten die in de gegevensbeschrijving (zie vorige paragraaf) gedefinieerd zijn. In het vorige hoofdstuk is een beschrijving opgenomen van G2 en voor een overzicht van de G2 syntax wordt de lezer verwezen naar bijlage D of [Gensym, 92]. Met behulp van de standaard G2-elementen zijn er daarnaast nog een aantal standaard SIX-elementen gedefinieerd die de gebruiker in staat stellen bepaalde redeneerstappen korter en makkelijker in te voeren. Het gaat hier om abstracte structuren die niet noodzakelijk zijn voor de domeinbeschrijving. De gebruiker kan hiermee de voorkeur volgorde voor het doseren van additieven meer vast leggen. Een aantal andere objecten komt later in deze paragraaf nog ter sprake. Het volgende hoofdstuk ("Implementatie") bevat de definitie van de verschillende objecten.

Met behulp van het G2 formalisme en de extra objecten kan de kennistechnoloog kennis over papier-fabricageproces en de problemen die zich kunnen voordoen invoeren in het systeem. Om te zorgen dat deze kennis handelbaar blijft is gekozen voor een aantal inferentiestappen:

- **initialisatie en classificatie** - Tijdens deze redeneerstap berekent het systeem een aantal variabelen die vooraf bepaald kunnen worden en gebruikt kunnen worden door de latere stappen. Daarnaast worden bepaalde door de gebruiker ingevoerde en berekende waarden geclassificeerd. Deze waarden krijgen dan een kwalitatieve waarde zoals: "erg laag", "laag", "hoog", etc. Na deze initialisatie en classificatiestap kunnen andere redeneerstappen gestart worden.
- **probleemdiagnose** - Deze stap signaleert een bepaald probleem in het proces. Er wordt dus een diagnose van het probleem gesteld. Met behulp van de kwalitatieve waarden die in de eerste stap bepaald zijn wordt bepaald welke afwijken van het optimale. Bepaalde afwijkingen of combinaties daarvan zijn signalen voor specifieke problemen in het proces.
- **advieskennis** - Als duidelijk is welk probleem er aanwezig is in het proces kan er op basis van de proceswaarden gekeken worden wat de beste oplossing is. Deze stap bepaalt naar aanleiding van een gesignaleerd probleem een advies voor de gebruiker.

Het is mogelijk dat bij uitbreiding van de functionaliteit van SIX het noodzakelijk blijkt dat er extra stappen toegevoegd moeten worden. Zoals als gezegd zouden stappen beschouwd kunnen worden als onderdelen waar een taaklaag gebruik van kan maken. In dit geval zou de taak dan "optimalisatie" zijn. Wanneer de functionaliteit van het systeem uitgebreid zou worden dan kan een taaklaag bepalen welke stappen er uitgevoerd moeten worden om een bepaald probleem op te lossen. De eerste stap zal in bijna alle gevallen de initialisatie- en classificatiestap zijn.

Initialisatie- en classificatiekennis

De initialisatie- en classificatiekennis bestaat uit alle objecten en regels die ervoor zorgen dat de eigenlijke redenering kan starten. Het gaat onder andere om omzetten van kwantitatieve waarden naar kwalitatieve waarden zodat de regels later daar gebruik van kunnen maken. Naast deze classificatiekennis wordt er een aantal van te voren te berekenen waarden bepaald zodat de andere redeneerstappen hier gebruik van kunnen maken.

Bepaalde ingevoerde meetwaarden en andere berekende waarden worden tijdens deze redeneerstap geclassificeerd zodat het voor de overige redeneerstappen eenvoudiger wordt om met deze waarden te redeneren. De meetwaarden worden geclassificeerd aan de hand van de interval-objecten. Deze objecten bevatten als attributen de naam van de te classificeren meetwaarde of variabele, een maximale waarde, een minimale waarde en een kwalitatieve waarde. Daarnaast bestaat er bijvoorbeeld voor de door de gebruiker ingevoerde pulp-meetwaarden een generieke regel die voor elke pulp-meetwaarde controleert of er een interval-object gedefinieerd is met een passend interval. Als dat zo is krijgt het attribuut kwalitatieve-waarde van het meetwaarde-object dezelfde waarde als de kwalitatieve waarde van het interval.

Het interval-object is voor de eindgebruiker of kennistechnoloog een uitbreiding op het G2 formalisme. Door nieuwe interval-objecten te maken worden bepaalde meetwaarden, eigenschappen of variabelen ook geclassificeerd tijdens deze stap. Later tijdens het redeneren kunnen de regels hier weer gebruik van maken en hoeft de regel niet alsnog zelf te controleren binnen welk interval een waarde valt. Bovendien is een regel duidelijker wanneer er gesproken wordt over een erg hoge cationic-demand dan wanneer er gesproken wordt over een cationic-demand met een waarde[13] tussen de 60 en 65.

Het classificatiemechanisme is tevens een gereedschap om het systeem te "tunen". Wanneer na verloop van tijd blijkt dat het systeem te snel een afwijking signaleert voor bijvoorbeeld de cationic-demand in het systeem kunnen de interval objecten voor cationic-demand aangepast worden. De regels zullen in de meeste gevallen geen wijzigingen te hoeven ondergaan. Het systeem is dus ook makkelijker onderhoudbaar door de classificatie.

Onder de vooraf te bepalen waarden vallen bijvoorbeeld de retentie van diverse stoffen (die ook geclassificeerd wordt) en of er wel of niet kationisch zetmeel toegevoegd is (kationisch systeem). Dit soort waarden zijn algemeen en kunnen door de latere redeneerstappen gebruikt worden. Het gaat hier in principe om vaststaande feiten zodra de gegevens over het systeem ingevoerd zijn.

Probleemdiagnose-kennis

Na de classificatiestap kunnen de afwijkingen in het proces (in de gemeten waarden) aangewezen worden en problemen in het proces worden gesignaleerd. Aan de hand van de rapporten van het Papierlab zijn enkele veel voorkomende problemen (oorzaken van de afwijkingen) in het papierfabricageproces onderzocht:

- dosering van een niet geschikt zetmeel
- toegevoegd zetmeel heeft onbedoeld de functie van trash-catcher
- doseringsplaats van een additief/zetmeel is niet optimaal
- doseringsvolgorde van additieven/zetmeel is niet optimaal
- ruimte in proces voor zetmeel niet optimaal benut
- overdosering van kationisch additief
- aanwezigheid van te veel anionische stoorstoffen
- aanwezigheid van bacteriële vervuiling

Twee problemen die op het eerste gezicht wel in het rijtje thuis horen zijn er niet in opgenómen: slechte retentie van het zetmeel en slechte retentie van de fines en fillers. Deze problemen bleken na analyse vaak terug te herleiden tot één of meer van de eerder genoemde problemen en zijn eigenlijk een gevolg (en een eigenschap van het proces aan de hand waarvan bepaalde problemen gesignaleerd kunnen worden).

Bovenstaande lijst is geen complete opsomming van de mogelijke problemen in het wet-end, maar een opsomming van de belangrijkste en meest voorkomende problemen. Om drie belangrijke redenen is besloten om de problemen waarover SIX kan redeneren te beperken tot bovenstaand lijstje:

---

[13] Deze waarden worden hier slechts als voorbeeld gebruikt.

1) Er moest vanwege de looptijd van het project een keuze gemaakt worden. Het is niet mogelijk om ineens alle mogelijke problemen te onderzoeken en er moest vastgesteld worden wat de mogelijkheden zouden zijn van SIX na afloop van het project.
2) Omdat deze problemen het meest voorkomen is er ook de meeste kennis te achterhalen zowel bij de experts als op papier waardoor het makkelijker is verbanden te leggen tussen de afwijkingen in het proces, de problemen en de oplossingen.
3) Het systeem heeft een voorbeeld/demonstratiefunctie voor AVEBE. Wanneer het systeem alleen de hele specifieke problemen kan oplossen en voor de meest voorkomende problemen geen oplossing heeft is het moeilijk aan te tonen dat het bouwen van zo'n systeem de moeite waard is.

Een nadeel van deze beperking is de ogenschijnlijke trivialiteit van de adviezen van het systeem. Dit is ook een nadeel voor de voorbeeldfunctie van het systeem en bij de presentatie van het resultaat naar AVEBE toe zal dan ook nadruk gelegd moeten worden op de uitbreidbaarheid van de kennis in het systeem. Het afgeleverde produkt zal slechts "basiskennis" bevatten. Overigens is dit een algemeen probleem bij het maken van een expertsysteem. Pas na langdurig gebruik en uitbreiding door experts zal een expertsysteem zijn waarde bewijzen.

De kennis die bij deze redeneerstap (voor de verschillende problemen) wordt gebruikt is opgeslagen door middel van variabelen of parameters, regels, functies en procedures. Bij voorkeur moet de redeneerkennis in regels opgeslagen worden. Deze zijn makkelijker te begrijpen omdat ze op een meer natuurlijke manier gedefinieerd zijn. In sommige gevallen kan het beter zijn om geen regels te gebruiken. Wanneer een regel erg diep genest en erg onoverzichtelijk wordt is het aan te raden een functie of procedure te definiëren. Daarnaast zijn er een bepaalde constructies in G2 die niet kunnen in een regel en waarvoor, jammer genoeg, een procedure gebruikt moet worden (hierover meer in het volgende hoofdstuk).

Tijdens de diagnosestap wordt een probleem gesignaleerd en gegevens over het probleem verzameld. Een goed voorbeeld van diagnose-kennis is de kennis die in SIX gebruikt wordt voor het constateren van een ongeschikt zetmeel dat toegevoegd is in het proces. Aan de hand van een aantal intervallen (zie vorige paragraaf) is een kwalitatieve waarde toegekend aan de belangrijke eigenschappen van een toegevoegd zetmeel. Op basis van de gemeten waarden uit het proces (en de kwalitatieve waarden daarvan) wordt gecontroleerd of het verband (regels uit de regelverzameling) tussen deze waarden klopt en of de eigenschappen van het zetmeel voldoen aan de verwachte waarden op basis van de gemeten waarden. Als dit niet het geval is worden de naam en functie van het zetmeel opgeslagen en moet er een "vlag" worden gezet dat er een verkeerd zetmeel wordt gedoseerd. Vervolgens is het aan de kennis in de advieslaag om op basis van de verzamelde gegevens over het probleem een goede vervanging te zoeken en via de interface de gebruiker te informeren.

De diagnose-regels worden na de classificatiestap geactiveerd waarna de regels ieder apart zullen reageren (of niet) op de waarden in hun condities. Deze stap bestaat voornamelijk uit forward-chaining. Alle invoergegevens zullen bekeken worden en zoveel mogelijk informatie zal gebruikt worden om zoveel mogelijk nieuwe informatie af te leiden en bepaalde vlaggen te zetten. Dit vormt weliswaar een doel, dus men zou kunnen zeggen dat het redeneren op basis van het doel gebeurt. Het mechanisme is echter forward-chaining omdat de invoergegevens het redeneren in deze stap op gang brengen. De vlag-variabelen vormen de overgang naar de volgende laag; de advieskennis.

Advieskennis

De kennis in deze stap bepaalt het advies voor de gebruiker. De regels worden gevuurd door de diagnose-stap wanneer er een probleem gesignaleerd is. Aan de hand van de geclassificeerde meetwaarden wordt vervolgens het advies bepaald. In bepaalde gevallen is het advies zodra de probleemdiagnose gesteld is triviaal. Bijvoorbeeld bij de constatering van overdosering van een bepaald additief ligt het advies erg voor de hand[14]. In andere gevallen moet er eerst een aantal berekeningen of regels doorlopen worden met behulp van de meetwaarden om een geschikt advies samen te stellen. Bij het voorbeeld van de dosering van het verkeerde zetmeel zal het systeem eerst nog een zetmeel moeten gaan zoeken met dezelfde functie en eigenschappen die het wel geschikt maken voor dit proces.

De advieskennis is onderverdeeld in dezelfde acht stukken als de diagnose-kennis. Elk stuk heeft een regel die vuurt wanneer een bepaalde diagnose-vlagvariabele een waarde krijgt. De advieskennis wordt dus in gang gezet door een forward-chaning mechanisme. Voor het geven van een advies wordt gebruik gemaakt van een

[14] Namelijk minder doseren.

"suggestie" object. Dit object bestaat uit een tekst (voor de beschrijving van de oplossing in natuurlijke taal) en een "geadviseerde-toevoeging". De tekst wordt gebruikt om het uiteindelijke advies via de interface module aan de eindgebruiker te geven. Het "geadviseerde-toevoeging" object bevat een hulpstof (additief of zetmeel) en een geadviseerde plaats en hoeveelheid. Dit object is vooral bedoeld om het advies niet alleen in tekst formaat op te hoeven slaan. Bij eventuele uitbreiding van het systeem waarbij er verder geredeneerd moet worden met het advies hoeft dit niet uit een tekst gefilterd te worden.

De suggestie wordt gevuld met gegevens die verzameld zijn de in de diagnose-fase (het tekstdeel) en gegevens verzameld tijdens deze adviesfase (het tekstdeel en de geadviseerde toevoeging). De conditie van de regel met de vlagvariabele bevat naast deze vlag ook de benodigde gegevens voor een bepaald advies in de conditie. De meeste van deze gegevens zullen op dat moment nog geen waarde hebben en het systeem zal door middel van het backward-chaining mechanisme proberen een waarde voor deze variabelen te achterhalen. Hierbij worden de overige regels (als nodig) uit de advieskennis gebruikt.

Wanneer deze laag de benodigde gegevens voor een advies verzameld heeft en er is een advies in natuurlijke taal geconstrueerd dan is het aan de interface om deze informatie aan de gebruiker te presenteren. De taak van de interface begint echter al eerder bij het opvragen van de meetwaarden en andere procesgegevens.

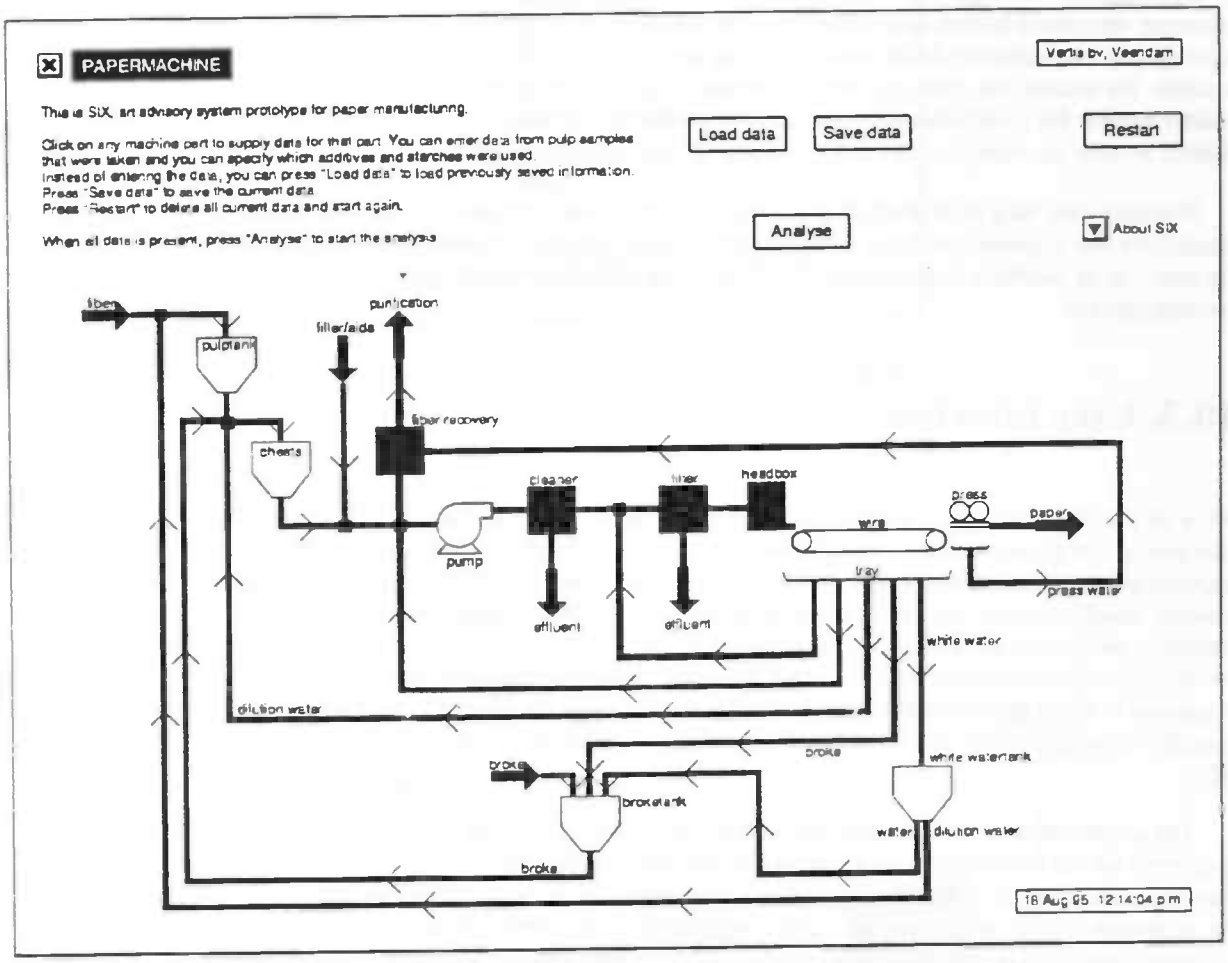
10.3. User interface

Voor de user interface is een schematische weergave van het wet-end van een papiermachine gekozen. Aangezien alle papiermachines verschillend zijn en we toch slechts één schema wilden opnemen, hebben we geprobeerd een zo algemeen mogelijk schema op te zetten, waarbij in ieder geval de belangrijkste onderdelen (chests, headbox, wire, tray, broke tank) herkenbaar waren. Het uitgangspunt voor deze opzet was de belevingswereld van de doelgroep (het Papierlab), waarvan we weten dat een schema van een papiermachine herkenbaar is en aanspreekt. Het schema is tot stand gekomen naar aanleiding van een ander project bij Vertis, waarvoor ook een papiermachine moest worden weergegeven (in G2 zelfs). Een aantal aanpassingen aan dat specifieke model hebben geleid tot het schema dat het hoofdscherm vormt van SIX en is weergegeven in figuur 10.2.

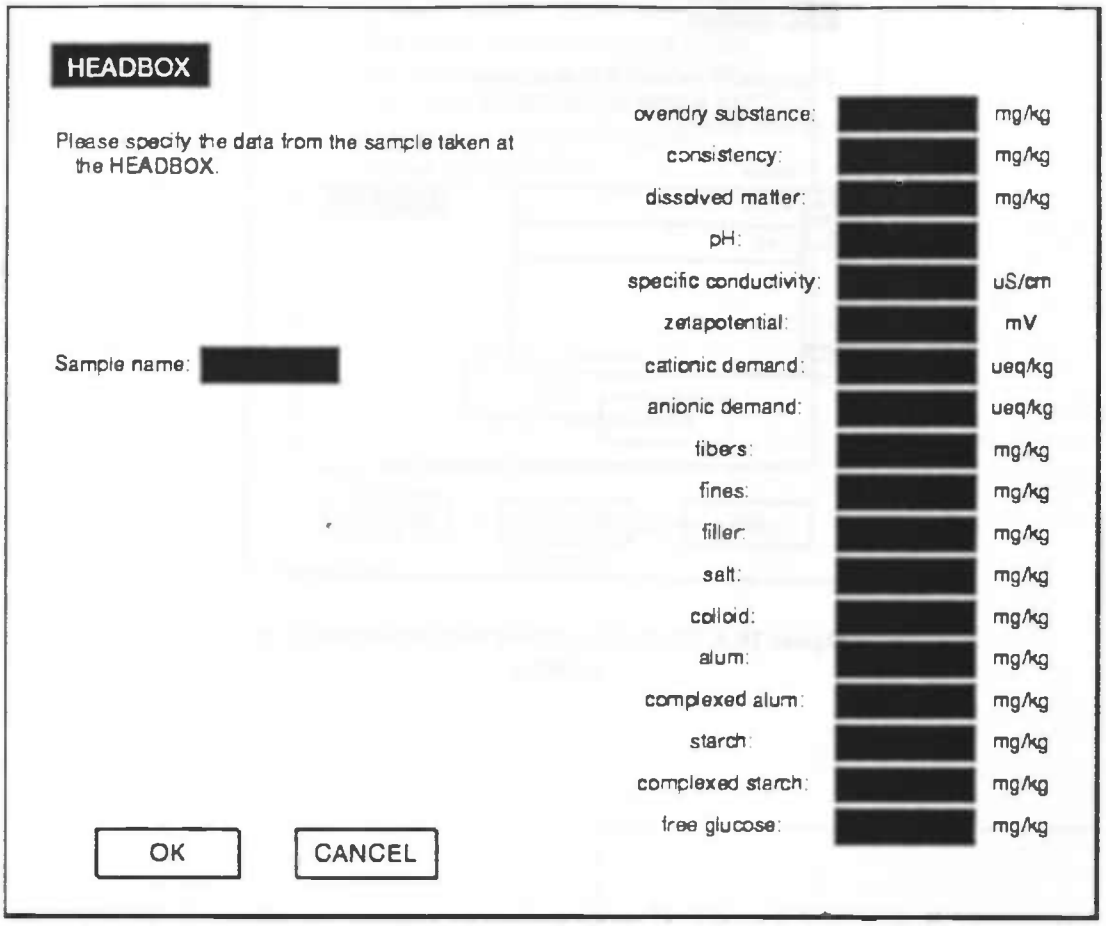
Het schema bestaat uit een aantal papiermachine-onderdelen. Voor elk van deze delen kan de gebruiker gegevens uit het fabricageproces invoeren. Er zijn drie soorten gegevens: gegevens van een pulpmonster dat op een bepaalde plaats in het proces is genomen, gegevens over de toevoeging van zetmeel op een bepaalde plaats en gegevens over de toevoeging van andere additieven op een bepaalde plaats. Deze opzet is realistisch; op één of meer plaatsen in de papiermachine worden pulpmonsters genomen en op een aantal plaatsen worden zetmeel en/of additieven toegevoegd. Meestal zijn er slechts twee of drie plaatsen in het proces van belang. De gebruiker hoeft alleen voor deze plaatsen gegevens in te voeren.

Door op een deel van de machine te klikken, verschijnt een menu met drie keuzes: "additive data", "sample data" en "starch data". Stel dat er meetgegevens van een pulp sample uit de headbox bekend zijn. Dan kunnen deze worden ingevoerd door op de headbox te klikken en "sample data" te selecteren. Dan zal het window verschijnen dat is weergegeven in figuur 10.3. De gebruiker kan een naam voor het pulpmonster invoeren en alle uit het monster afgeleide waarden. Als er in de chests (kuipen) een bepaald zetmeel is toegevoegd, kan met een klik op de chests het menu naar voren worden gehaald en "starch data" worden geselecteerd. Dan verschijnt het venster dat in figuur 10.4. staat. Uit de lijst kan een zetmeel worden gekozen en tevens kan de gebruiker aangeven hoeveel van dat zetmeel is toegevoegd in het proces. In figuur 10.5. is een soortgelijk invoervenster weergegeven, maar dan voor toegevoegde additieven. Als er een zetmeel is gebruikt, dat niet in de standaardlijst voorkomt, kan met de knop "NEW" (in figuur 10.4.) een zetmeel aan de kennis van SIX worden toegevoegd. Het dialoogvenster uit figuur 10.6. wordt hiervoor gebruikt. Voor het nieuwe zetmeel moet een categorie worden gekozen en moeten een naam en een degree of substitution worden ingevoerd. Verder moet de gebruiker aangeven of het zetmeel kationisch, anionisch of non-ionisch is. Er kunnen nog een aanbevolen dosering en een aanbevolen doseringsplaats worden opgegeven, maar dat is niet verplicht. Als bij het invoeren van additiefgegevens op de "NEW" button wordt gedrukt, verschijnt een venster dat net zo werkt als dat voor zetmeel, maar waarmee een nieuw additief aan het systeem kan worden toegevoegd.

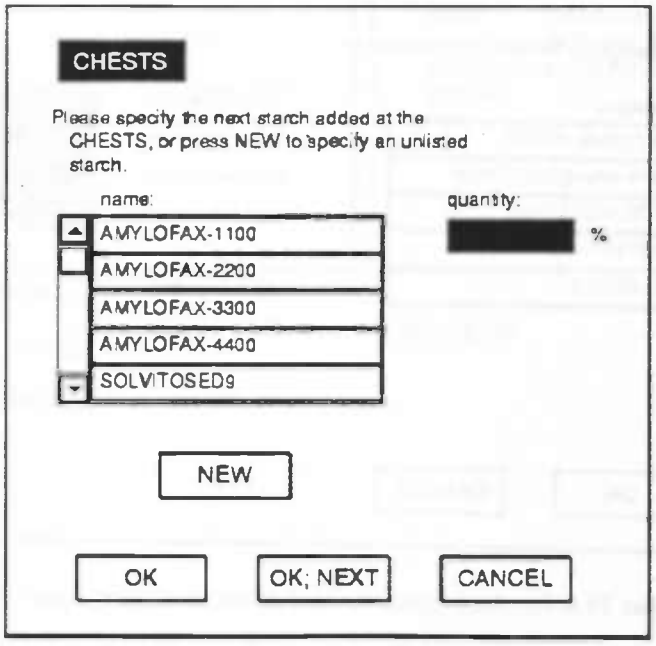
Als alle gegevens zijn ingevoerd, kunnen ze met de knop "Save data" worden bewaard en later weer met "Load data" worden opgevraagd (figuur 10.2.). De knop "Analyse" zet het redeneerproces van SIX in gang. Na het toepassen van de regels laat SIX een venster zien met de klassificatie van de gegevens, bijvoorbeeld zoals in figuur 10.7. Ten slotte zijn in figuur 10.8. nog twee voorbeeld-adviesvensters gegeven, waarin de gebruiker wordt geadviseerd alum te doseren en een ander zetmeel in het proces te gebruiken. Afhankelijk van de optimalisatiemogelijkheden zal SIX één of meer van dit soort vensters met advies tonen.



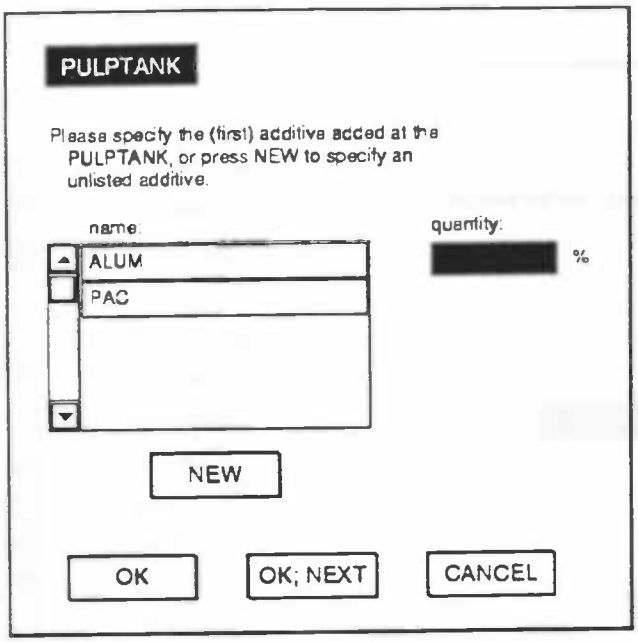
Figuur 10.2. Het hoofdvenster van SIX met een papiermachine-schema



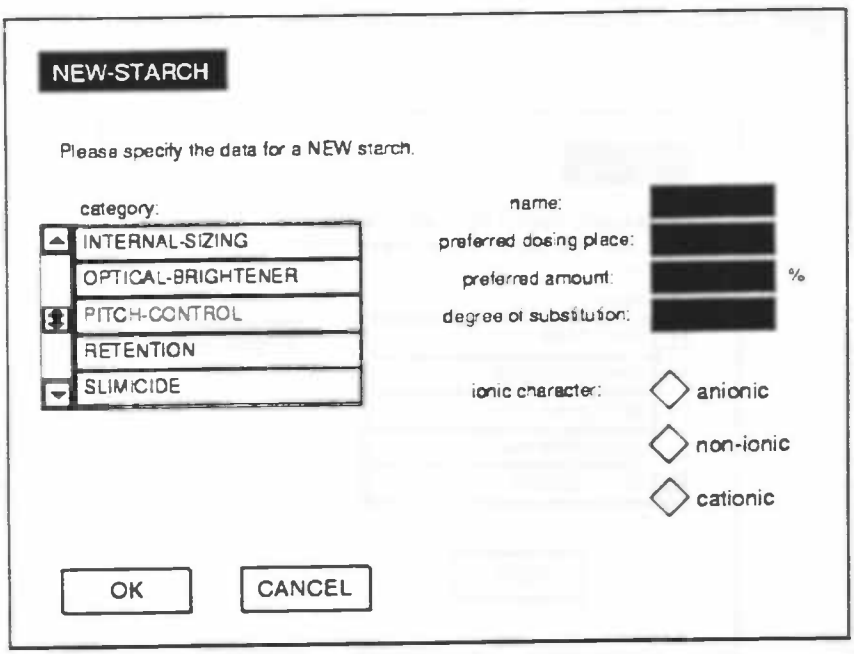
Figuur 10.3. Het dialoogvenster voor monsterwaarden gemeten in de headbox



Figuur 10.4. Het dialoogvenster voor zetmeeltoevoegingen in de chests



Figuur 10.5. Het dialoogvenster voor additieven in de pulptank



Figuur 10.6. Het dialoogvenster voor de invoer van een nieuw zetmeel

In the headbox I found that:
The specific conductivity is HIGH
The zeta-potential is NEGATIVE
The cationic demand is HIGH
The anionic demand is ZERO
The pH is NEUTRAL

OK

Figuur 10.7. Venster met klassificatie van gegevens

The amount of anionic trash in the proces
is HIGH (from measured CD).
To solve this problem you could try to dose alum
before dosing starches and retention aids.

OK

The degree of substitution of:
AMYLOFAX-1100 (D.S.: LOW)
doesn't match the specific conductivity
To correct this problem use another starch
instead:

Starch : AMYLOFAX-3300
(with d.s.: HIGH)
dosing-place : HEADBOX
amount : 0.0

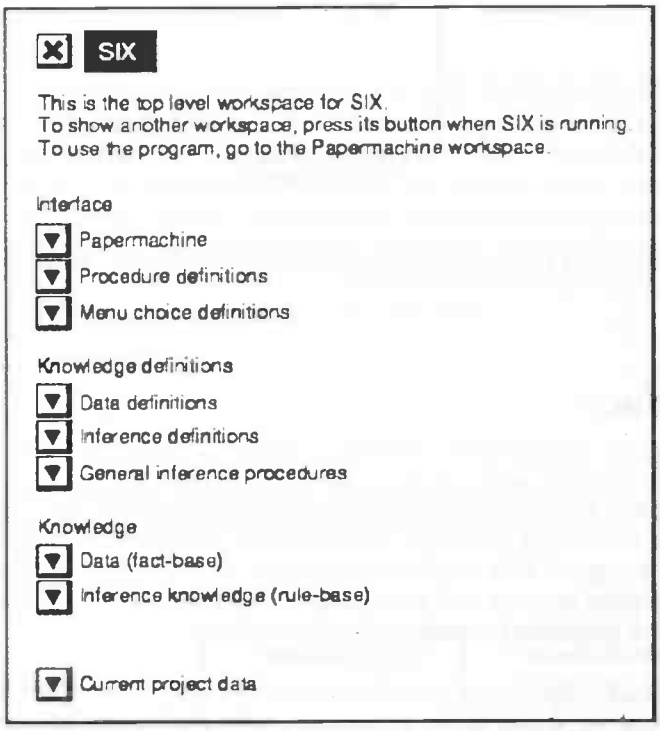
OK

Figuur 10.8. Vensters met advies

11. IMPLEMENTATIE SIX

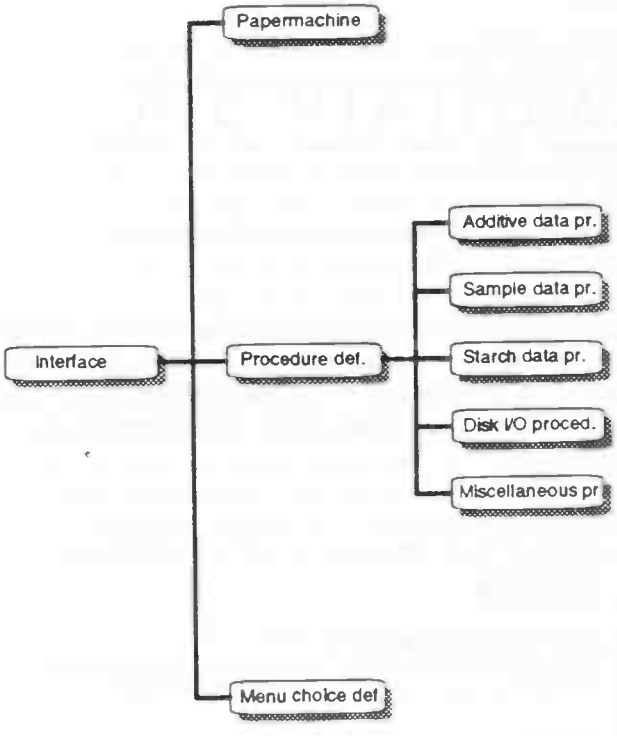
R.A. Carton § 10.1.; M. Pruis § 10.2.

Voor de implementatie van SIX is van G2 gebruik gemaakt. Dit ontwikkelsysteem bleek ondanks enkele tekortkomingen geschikt te zijn voor het onderhavige probleem. In G2 kan, zoals al in hoofdstuk 9 is gezegd, door middel van een hiërarchie van workspaces op een gestructureerde manier een applicatie worden opgebouwd. De belangrijkste workspace binnen SIX voor de ontwikkelaar is weergegeven in figuur 11.1.



Figuur 11.1. SIX top level workspace

Dit hoofdstuk is onderverdeeeld in twee delen waarin achtereenvolgens de implementatie van de interface en de implementatie van de kennis worden behandeld. Paragraaf 11.1. behandelt de boomstructuur die aanwezig is op de eerste drie workspaces van figuur 11.1. Paragraaf 11.2. beschrijft alle code in de boomstructuur van workspaces 4 t/m 8 uit het lijstje. Een uitzondering is gemaakt voor de workspace "papermachine" die zich in de boom onder "data definitions" bevindt. Deze workspace definieert weliswaar kennisobjecten, maar het belangrijkste aspect van de objecten op deze workspace is de grafische weergave (hieruit is de afbeelding van de papiermachine opgebouwd). Daarom is de bespreking van deze workspace in 11.1. opgenomen.



Figuur 11.2. De workspace-hiërarchie van de user interface

11.1. User interface

Figuur 11.2. geeft het deel van de workspace-hiërarchie weer dat belangrijk is voor de interface van SIX. De drie hoofd-workspaces die in de figuur staan zijn: Papermachine, Procedure definitions en Menu choice definitions. De eerste is in paragraaf 10.3. uitgebreid besproken, de tweede bevat verwijzingen naar alle interface-procedures en de derde bevat de drie menu-keuzes "Additive data", "Starch data" en "Sample data" (zie ook paragraaf 10.3.). De procedures worden hier nader bekeken.

Op de workspace genaamd Additive data procedures staan de procedures Get-additive-data, Additive-dialog en New-additive-dialog. Additive-dialog bouwt de dialoog uit figuur 10.5. op. New-additive-dialog zorgt voor een dialog zoals in figuur 10.6., maar dan voor additieven. Het ontvangen van de invoer wordt verzorgt door Get-additive-data en daarbij horen vier callback-procedures, te weten: Get-additive-data-OK-callback, Get-additive-data-cancel-callback, Get-additive-data-new-callback en Get-additive-data-next-callback. Deze procedures worden door G2 aangeroepen als de buttons van de Additive-dialog worden ingedrukt.

De workspace Starch data procedures herbergt procedures met een functionaliteit die sterk lijkt op die van de additive-procedures. De namen van de procedures zijn: Get-starch-data, Starch-dialog en New-starch-dialog. Er zijn ook vier callback-procedures, net als hierboven al is genoemd.

Tot de Sample data procedures behoort onder andere Get-sample-data. Deze zorgt voor de dialoog uit figuur 10.3. Bij de buttons horen Get-sample-data-OK-callback en Get-sample-data-cancel-callback. De teksten voor de invoervelden worden gegenereerd door Generate-sample-texts en de invoervelden (voorzien van een waarde als er al gegevens zijn ingevoerd op de bewuste plaats) worden in de dialoog gezet door Generate-sample-edit-boxes. Voor het werken met de invoerwaarden worden twee hulpprocedures gebruikt, namelijk: Set-value en Get-value-as-text.

Met de procedures van de Disk-I/O workspace kunnen gegevens worden bewaard en opgevraagd. Het bewaren gebeurt met Save-data, een procedure die gebruik maakt van Save-pulpsamples, Save-property, Save-additions en Save-starch-additions. Verder is er nog een procedure Save-error, die aan de gebruiker meldt dat er iets is misgegaan. Soortgelijke procedures maar dan voor het inladen van data zijn op deze workspace te vinden onder de namen Load-pulpsamples, Load-property, enzovoort.

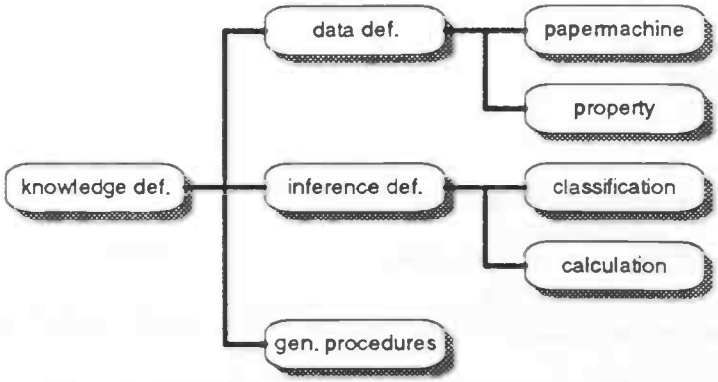
De laatste workspace heet Miscellaneous procedure definitions en bevat diverse soorten procedures. De procedure Call-the-experts is verbonden aan de Analyse-button en start het inferentieproces van SIX. Restart-SIX is verbonden aan de Restart-buttton en vraagt of de gegevens die zijn ingevoerd kunnen worden verwijderd (bij de procedure horen twee callbacks: Restart-SIX-yes-callback en Restart-SIX-no-callback). Save-current-project activeert Save-data, die al genoemd is. Close-dialog sluit een informatie-dialoogvenster als de gebruiker op OK drukt. Load-project (met Load-project-yes-callback en Load-project-no-callback). De procedures Give-general-advice en Give-specific-advice geven advies zoals wordt getoond in figuur 10.8. Bij het wegschrijven en inladen moet enige extra informatie worden verwerkt. Daarvoor zorgen Get-project-information en Get-project-filename (beide met een OK– en een cancel-callback). Drie hulpprocedures die worden gebruikt zijn: Get-float-from-text, Get-symbol-from-text en Remove-char-from-text.

11.2. Implementatie van de kennis

Zoals ook blijkt uit figuur 11.1. aan het begin van dit hoofdstuk is de kennis binnen SIX in de workspace-boom onderverdeeld in twee belangrijke delen: kennisdefinities (knowledge definitions) en kennis (knowledge). In figuur 11.3. is de onderliggende workspace structuur weergegeven. De eerste tak bevat definities, de zogenaamde gegevensbeschrijving (de implementatie van het E/R diagram uit het vorige hoofdstuk) en de algemene inferentie regels en objecten. Tevens zijn een aantal procedures die algemene gegevens berekenen of een algemene inferentiestap uitvoeren hier onder geplaatst. De tweede tak bevat voornamelijk de kennis zelf (gegevensverzameling & regelverzameling). In 11.2.1. wordt de inhoud van de eerste tak beschreven en wordt de objecthiërarchie toegelicht. De tweede tak komt in 11.2.2. aan bod.

11.2.1. Kennisdefinitie

De definitie van de kennisstructuur is zoals al gezegd opgedeeld in drie stukken: de zogenaamde gegevensbeschrijving, de algemene inferentie regels en objecten en verder een workspace met een aantal algemene procedures. In deze paragraaf wordt de inhoud en verdere onderverdeling van de workspaces besproken.



Figuur 11.3. Workspace hiërarchie "Knowledge Definition"

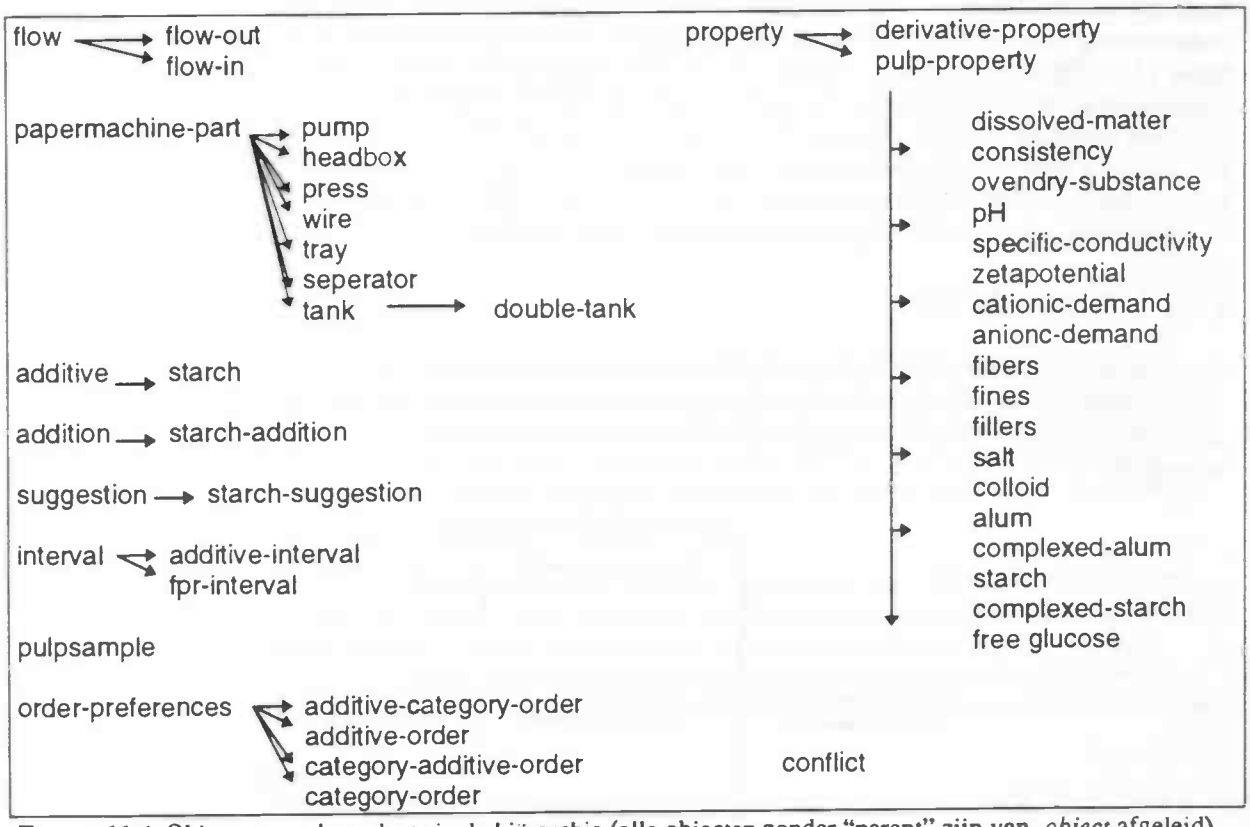
Data-definities

Het in hoofdstuk 9 beschreven E/R diagram kon, mede door het gedeeltelijk object georiënteerde karakter van G2, vrij direct geïmplementeerd worden. In figuur 11.4. is de objecthiërarchie schematisch weergegeven. Alle afgebeelde objecten zonder parent zijn afgeleid van het standaard G2 element *object*. Dit element bevat de

noodzakelijk gegevens voor G2 om met een object te kunnen werken maar verder geen specifieke attributen die voor SIX van betekenis zijn geweest.

De flow en papermachine-part objecten zijn al in paragraaf 11.1. besproken. Deze objecten bevatten een kleine hoeveelheid kennis over bijvoorbeeld hun onderlinge volgorde. Verder zijn ze voornamelijk van belang voor de interface.

Het property-object, en dan specifiek het pulp-property object is heel belangrijk voor SIX. Het pulp-sample-object maakt gebruik van deze objecten. In een pulp-sample-object worden alle meetwaarden die de gebruiker invoert opgeslagen. De subklassen van het pulp-property-object voegen geen extra attributen of eigenschappen toe. Het ouder object zelf bevat al een attribuut waarin de quantity of grootheid van de eigenschap opgeslagen wordt. Deze is voor de pulp-properties altijd gelijk aan de naam van de property. Dit lijkt een overbodige constructie, immers door op de quantity attribuut-waarde van een "pulp-property" te controleren is ook te achterhalen om welke eigenschap het gaat en de subklasses voegen niets toe. Er is op deze manier gewerkt met subklasses omdat dit een aanzienlijk voordeel oplevert bij het redeneren. In plaats van de constructie *for every pulp-property such that (specific-conductivity is ...)* te gebruiken kan in een regel de eenvoudiger structuur *for every specific-conductivity* gebruikt worden. De regels worden een stuk directer en de kennis wordt inzichtelijker. Verder lijkt de methode goed overeen te komen met het idee achter object georiënteerd programmeren. Het is beter om een structureel verschil tussen objecten (al is dat dan alleen in naam/betekenis en zijn de attributen gelijk) in de objecthiërarchie vast te leggen dan een extra attribuut te gebruiken om onderscheid te maken.



Figuur 11.4. Objecten met hun plaats in de hiërarchie (alle objecten zonder "parent" zijn van *object* afgeleid)

Het additive-object en subklasse starch zijn bedoeld om de informatie over hulpstoffen en zetmelen op te slaan. Deze objecten worden ook gebruikt in addition-objecten, om op te slaan wat er gedoseerd is. Wanneer de gebruiker een nieuw additief of zetmeel invoert, komt deze permanent in de knowledge-base te staan. In paragraaf 11.1. is beschreven hoe dit invoeren in zijn werk gaat. De definitie van een apart starch-addition-object is een noodoplossing voor een mankement in G2's objectgeoriënteerde karakter. Aanvankelijk was in het systeem alleen een addition gedefinieerd, met onder andere als attribuut een additive-object. Een starch-object is afgeleid van additive en een instantiatie van starch is dus volgens de standaard definities voor overerving in objectgeoriënteerde omgevingen een element uit de verzameling additieven. Het bleek in G2 niet mogelijk om aan een attribuut dat gedefinieerd is als additive-object een starch-object toe te kennen. Hierdoor is de definitie

van een starch-addition noodzakelijk, waardoor jammer genoeg ook sommige regels in het systeem weer minder inzichtelijk werden omdat ze dubbel uitgevoerd moesten worden.

Inference-definities

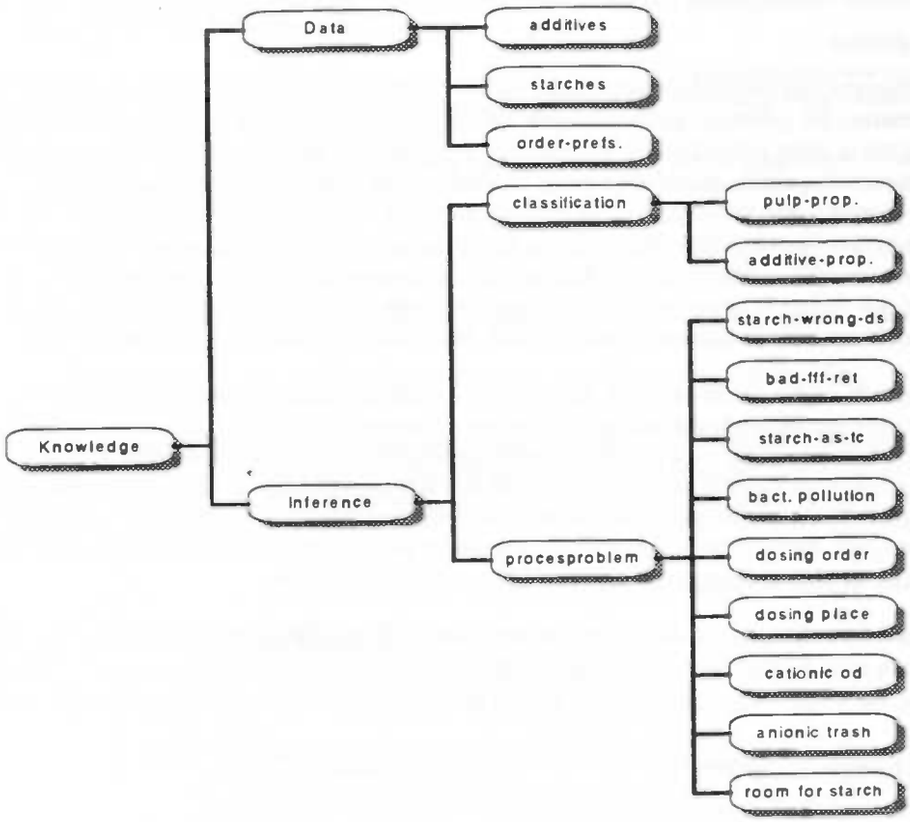
De overige objecten zijn bedoeld als hulpmiddel bij het redeneren en zijn daarom opgenomen bij de definities voor het redeneren. De gebruiker kan er bepaalde informatie in kwijt die het systeem vervolgens controleert. De interval-objecten worden gebruikt bij de classificatie. De gebruiker kan door middel van vier attributen (grootheid, minimale waarde, maximale waarde en kwalitatieve waarde) een kwalitatieve waarde vastleggen. Het systeem gebruikt deze vervolgens bij het classificeren. De order-preference-objecten zijn bedoeld om voorkeur doseringsvolgorden mee vast te leggen. Er kan twee additieven aangegeven worden welke er eerst gedoseerd moet worden, maar tevens kan deze informatie opgenomen worden over twee categorieën van additieven of een categorie en een specifiek additief. Tenslotte is er nog het conflict-object waar een resultaat van die volgorde controle in opgeslagen kan worden. Een conflict-object bevat twee additieven.

Het suggestion-object wordt door SIX gebruikt om het advies aan de gebruiker te presenteren. Het bestaat uit tekstlijst (voor het advies in natuurlijke taal) en een addition-object (of starch-addition) waarin diezelfde informatie nog eens bruikbaar voor het programma opgeslagen is. De adviestekst bevat wel uitleg waarom een bepaald advies gegeven wordt maar dit advies wordt vrij star gegenereerd en het is geen afleiding van de gebruikte regels. Op zich is dit wel te realiseren door in elke regel een extra actie in de consequent op te nemen die ergens bewaard dat de regel gevuurd heeft en wanneer precies. Dit komt de inzichtelijkheid van de regels niet te goede en daarom is besloten dit niet in te bouwen in het prototype.

Hele belangrijke definities in dit deel zijn de algemene regels op de classificatie-workspace. Deze zorgen met behulp van door de kennis-engineer ingevoerde intervallen (workspace knowledge→ inference→ classification, zie volgende paragraaf) dat de tijdens een sessie ingevoerde meetwaarden geclassificeerd worden.

11.2.2. Kennisverzameling

De kennisverzameling is gestructureerd door middel van onderstaande workspaces. Deze structuur bevat de kennis die bij het begin van een sessie aanwezig is. Alle zetmelen die het systeem kent zijn bijvoorbeeld opgeslagen op de starches workspace.



Figuur 11.5. Workspace hiërarchie "Knowledge"

Gegevens- of feitenverzameling

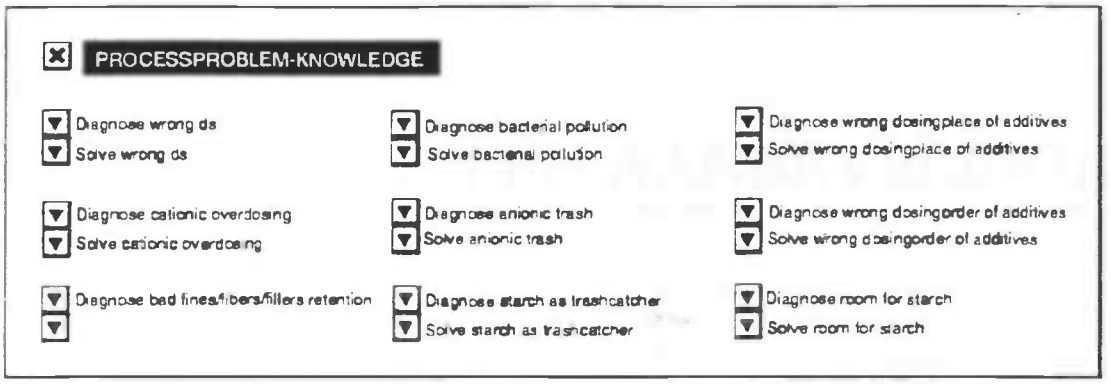
De gegevens of feitenverzameling is onderverdeeld in twee soorten kennis. De kennis die al aanwezig is voordat de sessie begint en de kennis die tijdens de sessie ingevoerd wordt. Een deel van die tweede soort kan een volgende keer deel zijn van de eerste soort. Hieronder vallen nieuw toegevoegde zetmelen en andere hulpstoffen. De kennis die sessie afhankelijk wordt opgeslagen op een daarvoor speciaal gedefinieerde workspace "current-project-data". Deze valt dus niet onder de hierboven besproken objecthiërarchie en bevat alle informatie van de sessie die op een bepaald moment gaande is.

De sessie onafhankelijke (niet in regels opgeslagen) kennis is opgenomen in de afgebeelde data, starches, additives en order-preferences workspaces. Verder zijn op de classification→ pulp-property en classification→ starch-property die specifieke redeneer objecten (voornamelijk interval-objecten) opgeslagen. Voor alle duidelijkheid: alle hier opgenomen kennis bestaat uit instantiaties van de in de vorige paragraaf besproken object-definities. Deze zijn door de kennis-engineer ingevoerd of tijdens een sessie door de gebruiker ingevoerd.

Regelverzameling

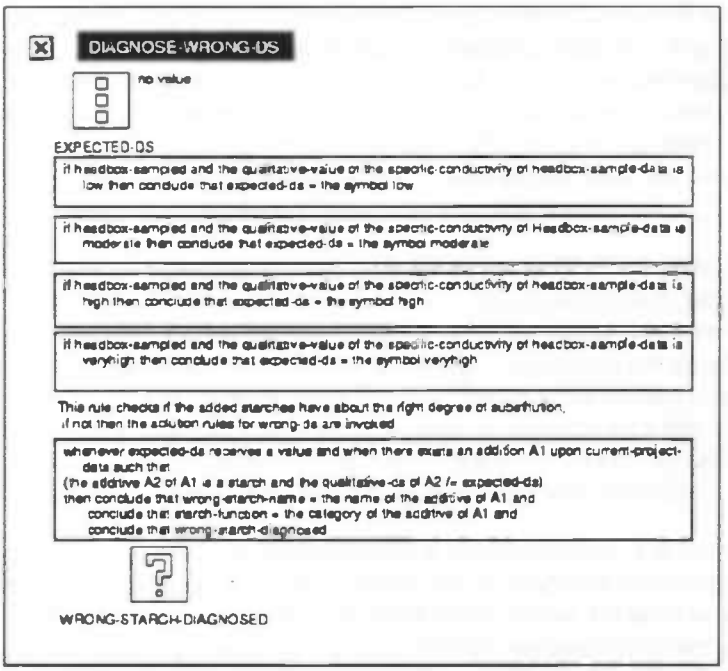
De kennis in de regelverzameling is onderverdeeld volgens de lagenstructuur zoals die ook in hoofdstuk 9 behandeld is. De kennis bestaat uit: classificatie– en initialisatie-kennis, diagnose-kennis en advieskennis. De classificatiekennis is opgenomen in de eigen workspace-hiërarchie.

De diagnose– en advieskennis zijn in **één** workspace-hiërarchie opgenomen. De reden voor het combineren van deze lagen is de onderverdeling naar mogelijke problemen. Deze onderverdeling is wel gehandhaafd gebleven. De kennis over elk probleem is wel weer onder verdeeld naar diagnose– en advieskennis. Figuur 11.6. is de "processproblem-knowledge" workspace. Vanuit deze workspace kan de kennis-engineer de andere workspaces voor diagnose en advies van de diverse problemen bereiken.



Figuur 11.6. De "procesproblem-knowledge" workspace, regelverzameling

Figuur 11.7. is een voorbeeld van een diagnose-workspace. Op basis van een aantal regels en kenmerken wordt een probleem gesignaleerd; in dit geval het gebruik van een verkeerd zetmeel. De benodigde gegevens worden verzameld en opgeslagen. Tegelijkertijd wordt de "vlag"-variabele gezet. In dit geval is dat wrong-starch-diagnosed.



Figuur 11.7. De "diagnose-wrong-ds" workspace

De "vlag"-variabele zorgt er door het forward-chaining mechanisme van G2 voor dat de advieskennis voor dit specifieke probleem meteen in actie komt. De advieskennis is ook weer op een eigen workspace ingedeeld. Daarop bevinden zich soortgelijke regels, variabelen en eventueel procedures en functies. Het gebruik van procedures en functies in diagnose- en advieskennis is zoveel mogelijk vermeden. Het nadeel van vooral procedures is dat in de regel waarin de procedure aangeroepen wordt niet duidelijk is welke variabelen gemanipuleerd worden. Er is dus, door alleen de regel te bekijken, niks te zeggen over welke andere regels als gevolg hiervan aangeroepen zouden kunnen worden. Op sommige momenten was het gebruik van procedures echter noodzakelijk omdat een bepaalde reeks acties niet in een regel opgenomen kon worden of alleen over veel, betekenisloze, regels verdeeld kon worden. Betekenisloos omdat de regels dan een actie zouden uitvoeren en vervolgens een dummy-variabele zetten, zodat een andere regel met de volgende actie aangeroepen zou worden. Het gebruik van een procedure was in een aantal gevallen duidelijker dan het verspreiden van de acties over veel regels.

12. RESULTATEN EN AANBEVELINGEN

Er kan worden gesteld dat met het prototype SIX de in het tweede hoofdstuk gestelde doelstelling gehaald is. Het resultaat bestaat uit een visueel aantrekkelijk systeem dat eenvoudig te bedienen is en een redelijke hoeveelheid kennis over het papierfabricageproces bevat. SIX geeft op kwalitatief niveau advies voor het verbeteren van het proces. Het advies heeft betrekking op de dosering van zetmeel en andere hulpstoffen in het wet-end. Met het programma laat Vertis aan AVEBE een toepassing zien van expertsystemen in de procesindustrie. Vertis heeft hiermee tevens de beschikking gekregen over een in G2 gerealiseerd voorbeeld van een regelgebaseerd expertsysteem.

Een probleem van SIX in zijn huidige vorm zou het, in de ogen van de experts, triviale karakter van de gegenereerde adviezen kunnen zijn. Dit is een probleem dat bij veel expertsystemen in de aanloopfase gesignaleerd wordt. Afhankelijk van het gebruik bij AVEBE en ondersteuning door Vertis zal het systeem zich in de toekomst moeten bewijzen als een in de praktijk (bijvoorbeeld voor vertegenwoordigers) bruikbare applicatie. De kennis is op gestructureerde en inzichtelijke wijze aanwezig in het systeem waardoor het door iemand met G2-ervaring vrij gemakkelijk kan worden uitgebreid. Wel verdient het aanbeveling voordat de applicatie in praktijk gebruikt gaat worden een kennis-editor te ontwikkelen waarmee een expert van het Papierlab zelf kennis kan invoeren.

Tijdens de ontwerpfase bleek dat het standaard niet mogelijk is om in G2 de afleiding te geven die tot een bepaalde conclusie heeft geleid. Door deze tekortkoming kan SIX dus geen overzicht geven van de regels die zijn gebruikt om tot een bepaald advies te komen. Het systeem geeft dus wel adviezen en kan deze ook (beknopt) beargumenteren, maar niet in de vorm van een afleiding. Overigens is het mogelijk om SIX aan te passen, zodat er wel afleidingen worden gegeven, maar dan moet iedere regel worden uitgebreid met tenminste één actie. Om de regelverzameling inzichtelijk te houden, hebben we besloten dit achterwege te laten. G2 zou standaard in staat moeten zijn om een overzicht te geven van de regels die tot een bepaalde conclusie hebben geleid. Hopelijk kunnen toekomstige versies van G2 dit soort informatie wèl verstrekken.

Een ander nadeel van het programma is de slechte feedback van de interface. Dit is echter een platformafhankelijk probleem, dat inherent is aan het gebruik van het grafische ontwikkelgereedschap GUIDE/UIL van Gensym. Het overstappen naar een nieuwere versie (versie 4.0 heeft nu de beta-status) van G2 en GUIDE/UIL en de daarvoor benodigde aanschaf van een snellere computer met meer geheugen zouden deze situatie aanzienlijk verbeteren. Versie 3.0 maakt (in vergelijking met de interfaces van bijvoorbeeld X Windows en Windows NT) op sommige punten een gedateerde indruk. In de nieuwe versie is een aantal verbeteringen op het gebied van de bediening doorgevoerd; deze wijzigingen zorgen ook voor een moderner uiterlijk van de applicaties.

Het prototype SIX is momenteel toegespitst op het papierfabricageproces. De structuur van de programmacode is vrij specifiek voor het hier beschreven probleem en datzelfde geldt voor de structuur van de kennis. Hierdoor is het voor een leek op het gebied van G2 niet eenvoudig deze structuren te wijzigen of uit te breiden. Als SIX in de praktijk gebruikt gaat worden, is het wenselijk dat zowel de code als de kennis veralgemeniseerd worden. De duur van onze afstudeerperiode stond dit echter niet toe (bovendien valt het niet onder de opdracht tot het ontwerpen van een prototype). Het formaliseren van SIX heeft een aantal voordelen, te weten:

- Het programma kan makkelijker worden aangepast voor bijvoorbeeld de textielindustrie of de levensmiddelenindustrie, wanneer de daar gebruikte kennis daarvoor geschikt wordt geacht (de hoeveelheid beschikbare kennis over de papierfabricage en in het bijzonder over het wet-end is groter dan de

hoeveelheid kennis uit de twee andere industrietakken en bovendien is deze kennis door AVEBE èn Vertis makkelijker expliciet te maken in de vorm van regels en formules).

- Het ontwikkelen en het onderhouden van een kennis-editor is makkelijker omdat er sprake is van een geformaliseerde kennisstructuur. Met een kennis-editor kunnen nieuwe regels op relatief eenvoudige wijze in het systeem worden aangebracht, omdat de editor zorgt voor de representatie ervan.
- De kennis van SIX kan beter worden uitgebreid en aangepast, zelfs door mensen met weinig G2-ervaring, omdat gebruik moet worden gemaakt van de kennis-editor.

Een andere interessante optie voor SIX is het uitbreiden tot een real-time expertsysteem door een koppeling tot stand te brengen met het papierfabricageproces. Het programma kan dan de gegevens van het proces zelf verzamelen. Eventueel kan als tussenstap (en later als ondersteunende functionaliteit) een koppeling met een model van het papierfabricageproces tot stand gebracht worden. Binnen een ander project probeert Vertis momenteel een papierfabriek te modelleren. Het wiskundig model zou dan gebruikt kunnen worden om acties van de operator door te rekenen zodat de effecten in het echte proces voorspeld kunnen worden. Tot nu toe heeft Vertis voornamelijk van G2 gebruik gemaakt als interface naar verschillende processen. SIX gebruikt de mogelijkheden van de expert-shell van G2. Een koppeling tussen het fabricageproces en een expertsysteem (intelligent control) zou optimaal gebruik maken van de krachtige real-time redeneermogelijkheden van G2. Opgemerkt moet worden dat een dergelijk systeem gezien de hoeveelheid kennis die op dit moment geformaliseerd is en de stand van zaken wat betreft een wiskundig model de komende jaren wel toekomstmuziek zal blijven. Desalniettemin is dit een interessante doelstelling voor de verdere ontwikkeling van SIX.

BIJLAGE A: WOORDENLIJST

additief	Chemische toevoeging of hulpstof.
AI (artificial intelligence)	Zie kunstmatige intelligentie.
automatic explanation	Het laten verklaren van gebeurtenissen of verbanden door computers.
backward reasoning	Redeneren vanuit het doel (of de conclusie) naar de initiële gegevens (of argumenten).
beslisbaar (logica)	Als van een theorie in eindige tijd bepaald kan worden of deze bewijsbaar is of niet.
beslisboom	Een structuur waarin alle m ogelijke inferentiestappen vanuit een bepaalde situatie zijn opgenomen.
best-first zoekmethode	Een combinatie van de depth-first en breadth-first zoek-methoden, waarbij steeds het meest veelbelovende zoekpad het eerst wordt onderzocht.
certainty factor	Zekerheidsfactor; een waarde die aangeeft in hoeverre zeker is dat een bepaald feit waar is.
check box	Aankruisvak; hiermee kunnen in programma's opties worden aan- en uitgezet.
conflict set	Verzameling mogelijk toepasbare (relevante) regels.
conflictresolutie	Methode om uit de conflictset een regel te kiezen om toe te passen.
database	Geordende verzameling gegevens.
derivatisering	Bewerking van de chemische structuur van een natief zetmeel.
direct manipulation interface	Interface-type waarin de gebruiker zijn gegevens direct kan manipuleren. Door ikonen op het scherm te verplaatsen, worden de daarbij behorende bestanden gekopieerd of weggegooid.
E/R diagram	Entiteit/Relatie-diagram; schema dat entiteiten en de relaties daartussen weergeeft.
eenwaardige variabele	Variabele die slechts één waarde kan aannemen.
entiteit	Grootheid; opzichzelfstaand object.
event	Gebeurtenis; een wijziging in een deel van een systeem die effect kan hebben op andere delen uit dat systeem.
expertsysteem	Systeem dat adviezen geeft op basis van kennis van menselijke experts.
expertsysteem-shell	Omgeving waarin op eenvoudiger wijze dan in de traditionele omgevingen een expertsysteem ontwikkeld kan worden.
formalisme	Nauwkeurig omschreven manier om zaken weer te geven of te structureren.
forward reasoning	Redeneermethode vanuit de intiële gegevens waarbij getracht wordt zoveel mogelijk nieuwe kennis (en doelen) af te leiden.
front-end	Het deel van een systeem of programma waarmee de gebruiker of een ander systeem te maken heeft.
fuzzy logic	Vorm van logica waarin niet met TRUE en FALSE wordt geredeneerd, maar met linguïstische variabelen, die waarden zoals "zeer hoog" en "licht afwijkend" kunnen aannemen.

heuristisch	Met niet-geformaliseerde beoordelingskennis.
hybride	Gecombineerd.
implementatie	Realisatie van een systeem in een bepaalde programmeertaal of omgeving.
interpreter	Systeem dat stapsgewijs programma's vertaalt en uitvoert.
iteratief	Stapsgewijs.
kennisacquisitie	Het verzamelen van kennis (feiten en regels) volgens een bepaalde techniek, bijvoorbeeld via interviews.
kennis-editor	Programma waarmee de kennis in een expertsysteem op relatief eenvoudige wijze kan worden bewerkt en uitgebreid. De editor bewaakt de consistentie van de kennis.
kennisrepresentatie	Manier waarop kennis gestructureerd en opgeslagen wordt.
kennissysteem (in deze scriptie)	Zie expertsysteem.
klantspecificatie (van de klant)	Wensen van de klant betreffende het papier.
knowledge base	Gegevensverzameling en regelverzameling. De kennis in een expertsysteem.
kunstmatige intelligentie	"De studie naar manieren om computers taken te laten uitvoeren waarin de mens op dit moment beter is." [Rich, 91]
literal	Constante.
meerwaardige variabele	Variabele die meerdere waarden kan hebben.
message window	Afgebakend deel van het scherm waarop een boodschap aan de gebruiker wordt getoond.
methode	Procedure of functie die onderdeel is van een object in een objectgeoriënteerde omgeving.
natpartij	Deel van de papierfabricage waar pulp wordt omgezet in papier.
natuurlijke taal	De klasse van talen zoals we die in het dagelijks leven gebruiken.
neuraal netwerk	Systeem dat de werking van de hersenen nabootst, door gebruik te maken van een stelsel van onderling verbonden neuronen, die elkaar beïnvloeden.
objectgeoriënteerd (programmeren)	Programmeren uitgaande van objecten en hun eigenschappen. Een object bestaat uit programmacode en gegevens.
off-line	Niet met een computer of een ander systeem verbonden.
on-line	Met een computer of een ander systeem verbonden.
onbeslisbaar (logica)	Als van een theorie niet in eindige tijd bepaald kan worden of deze wel of niet bewijsbaar is.
oploopkast	Headbox; deel van de papiermachine dat zorgt voor de vorming van een vel papier uit pulp.
papiereigenschap	Eigenschap van het papier; bijvoorbeeld interne sterkte.
pattern matching	Het bij elkaar zoeken van patronen met dezelfde kenmerken.
pilot project	Experimenteel project; eerste aanzet.
proceseigenschap	Eigenschap van een proces. Bijvoorbeeld pH of zetapotentiaal.
produktieregel	Structuur waarmee vanuit een bepaalde conditie of antecedent een consequent (of actie) kan worden afgeleid.
pulp	Mengsel van water, houtvezels, vezels van oud papier en chemicaliën.
pulp sample	Monster van de pulp, genomen op een bepaalde plaats in de papiermachine.
radio button	Hiermee kan in een programma één optie uit een aantal opties worden gekozen.
real-time	In echte tijdseenheden; op het moment.
resolutie (oplossend vermogen)	Aantal pixels waaruit het scherm is opgebouwd.
retentie	Opname.
semi-beslisbaar (logica)	Als van een theorie in eindige tijd bepaald kan worden dat deze bewijsbaar is maar niet gegarandeerd in eindige tijd bepaald kan worden dat deze niet bewijsbaar is.

SIX	Starch Intelligent Expert (oorspronkelijke betekenis; is nu niet meer van kracht).
statement	Opdracht; commando.
substitutie	Systematische vervanging van symbolen (door andere symbolen).
time-tag	Variabele die de tijd bevat waarop een bepaald (gekoppeld) feit afgeleid is.
traceren	Het afleidingspad bepalen.
unificatie	Het bij elkaar zoeken van twee variabelen omdat ze dezelfde waarde hebben of omdat één van beide nog geen waarde heeft.
user interface	Het deel van een programma dat de gebruiker ziet en waarmee hij werkt.
variabele	Symbolische representatie voor een waarde (bijvoorbeeld "x" in plaats van een getal).
wet-end	Zie natpartij.
zetmeelderivaat	Natief zetmeel dat een bepaalde chemische bewerking heeft ondergaan.

BIJLAGE B: INTERVIEWS

In deze bijlage zijn de uitwerkingen opgenomen van de interviews met de experts (Jan Hendriks en Jacob Terpstra) van het AVEBE-Papierlab. Met deze mensen hebben sinds de start van het project vijf interviews plaatsgevonden. Wegens het vertrouwelijke karakter is deze informatie alleen in de exemplaren voor Vertis en AVEBE aanwezig.

BIJLAGE C: VOORBEELDRAPPORT PAPIERLAB

Wegens het vertrouwelijke karakter is deze informatie alleen in de exemplaren voor Vertis en AVEBE aanwezig.

BIJLAGE D: G2 LANGUAGE REFERENCE

De language reference van G2 geeft een beknopt maar compleet overzicht van de taal G2.

G2 Language Reference

Functions

Arithmetic Functions

abs(<expression>)
arctan(<expression-1> [,<expression-2>])
average(<expression-1>, <expression-2> [,...])
ceiling(<expression>)
cos(<expression>)
exp(<expression>)
expt(<expression-1>, <expression-2>)
floor(<expression>)
ln(<expression>)
log(<expression>)
max(<expression-1>, <expression-2> [,...])
min(<expression-1>, <expression-2> [,...])
quotient(<expression-1>, <expression-2>)
random(<expression-1> [,<expression-2>])
remainder(<expression-1>, <expression-2>)
round(<expression>)
sin(<expression>)
sqrt(<expression>)
tan(<expression>)
truncate(<expression>)
truth-value(<expression>)

Call Function

call-function (<function-definition> [,arguments])

Connection Evaluator Functions

connection-direction (object-reference, connection-reference) = (symbol)
connection-portname (object-reference, connection-reference) = (symbol)
connection-position (object-reference, connection-reference) = (integer)
connection-side (object-reference, connection-reference) = (symbol)
connection-style (object-reference, connection-reference) = (symbol)

Foreign Functions

Refer to the Functions chapter.

Quantity Function

quantity ((<logical-expression> | <text>)) = (quantity)

Symbol Function

symbol (<text-expression>) = (symbol)

Tabular Functions of One Argument

Refer to the Functions chapter.

Text Functions

capitalize-words (<text>)
get-from-text (<text>, <position-1>, <position-2>)
insert-in-text (<text-1>, <text-2>, <position>)

Text Functions continued

is-contained-in-text (<text-1>, <text-2>)
length-of-text (<text>)
lower-case-text (<text>)
omit-from-text (<text>, <position-1>, <position-2>)
position-of-text (<text-1>, <text-2>)
text-begins-with-quantity (<text>)
upper-case-text (<text>)

Time Functions

year (<time>)
month (<time>)
day-of-the-month (<time>)
day-of-the-week (<time>)
minute (<time>)
hour (<time>)
second (<time>)
time (<year>, <month>, <day>, <hour>, <minute>, <second>)

User-Defined Functions

<function name> ((<argument-1> [, <argument-2> ...])) = <expression>

Syntax Key

< >
Indicates something that you supply.

[ ]
Indicates that something is optional, except when bold (indicates that brackets are part of the syntax).

{choice1 | choice2}
Indicates a choice. Choices are separated by a bar.

[, ...]
Indicates that you can repeat a specification any number of times.

Actions

Abort

abort (<procedure> | <procedure-invocation>)

Activate

activate the subworkspace of <object>

Change

change (<g2-array> | <g2-list>) [<integer-expression>] = <item-or-value>
change the array-length of <g2-array> to <expression>
change the <color-attribute> of <item> to <color-expression>
change the color-pattern of <item> so that <color-attribute> = <color-expression> [, <color-attribute> = <color-expression>...]
change the name of <item> to <symbolic-expression>
change the size of <workspace-expression> to minimum
change the text of the <attribute> of <item> to <text-expression>
change the text of [<procedure> | <statement> | <free-text> | <message>] to <text-expression>

Conclude

conclude that [<g2-array> | <g2-array>] [<integer-expression>] [= <expression> | is <symbol>]
conclude that <attribute> of <object> [= <expression> | is <symbol>]
conclude that the <class-or-data-type> that is an attribute of <item> named by <symbolic-expression> [= <expression> | is <symbol>]
conclude that [not] <logical-var-or-param> [[is true | is false | = <logical-expression>]]
conclude that <symbolic-var-or-param> [is <symbol> | = <symbolic-expr>]
conclude that <quantitative-var-or-param> = <arithmetic-expression>
conclude that <text-var-or-param> = <text-expression>
conclude that <variable> has no [current] value
conclude that [<item> | every <class>] is [[not | now]] <relation-to> [<item> | every <class>]

Create

create [(a | an) <class> [by cloning <item>] | an instance of the class named by <symbolic-expression>]
create a connection [of [class <connection> | the class named by <connection expr>]] connected [<between-spec> | <to-spec>] <connection-attr>[, ...]

Deactivate

deactivate the subworkspace of <object>

Delete

delete [<item> | <connection-reference> [removing connection stubs]]

Focus

focus on <object>

Halt

halt [with <text-message>] if breakpoints are enabled

Hide

hide [the workspace of <item> | the item superior to <item> | the subworkspace of <item> | this workspace] [on <g2-window>]

Inform

inform [the operator | <object>] [on <workspace [below <item> | above <item>]] [for the next <time-interval>] that <text-message>

Insert

insert <new-element> [at the [beginning | end] of | [before | after] <element> in] the <list-type> list <list>

Invoke

invoke <rules-category> rules [for <object>]

Make

make <item> permanent
make <item> transient
make <kb-workspace> the subworkspace of <item>

Move

move <icon> [to | by] (<x-workspace-units-right>, <y-workspace-units-up>)

Pause

pause knowledge-base

Print

print <kb-workspace>

Remove

remove [<element> from the <list-type> list <list> | the [first | last] <list-type> from <list>]

Reset

reset knowledge-base

Rotate

rotate <icon> [by | to the heading] [0 | 90 | 180 | 270] degrees

Set

set <gsi-variable> to [<quantitative-expression> | <logical-expression> | <symbolic-expression>]
set <simulation-variable> to <quantitative-expression>

Show

show <workspace-or-item> [on <g2-window> | at <scale>] [with its <item-location> at [<integer> units | the <screen-location> of the screen]]

Shut down G2

shut down g2

Start

start <procedure> [<argument(s)> [across <g2-to-g2-interface>] [at priority <integer>] [after <time-interval>]]

Transfer

transfer <item> [to <kb-workspace> [at [<x-coordinate>, <y-coordinate>]] | off]

Update

update <display>

Procedure Statements

Assignment

<local name> = <expression>

Call

[x, ... =] call <procedure> [<arguments>] [across <g2-to-g2-interface>]

Case

case [<expression>] of
<case-tag>: <statement>; ...
[otherwise:
<statement>[; ...]]
end

Collect Data

collect data [[timing out after <time-interval>]]
<local name> = <expression> [; ...]
[; if timeout then <statement>][;]
end

Do in Parallel

do in parallel [until one completes]
<statement>[; ...]
end

Exit

exit if <logical-expr>

For

for <local name> = each <item-or-value-ref>
do [in parallel [until one completes]]
<statement> [; ...] end

for <counter> = <numeric-expr-1> [to | down to]
<numeric-expr-2> [by increment]
do [in parallel [until one completes]]
<statement> [; ...] end

Procedure Syntax

<proc-name> ([<local name>:<type>, ...])
[= ([<type>, ...])]

[<local name>:<type> = <expression>;]
[<local name(s)>:<type>;]
begin
<statement(s)>;
end

Go To

go to [<symbol> | <integer>]

If-Then

if <logical-expression> then <statement>[else <statement>]

On Error

on error [<error-name>, <error-text>] [<statement(s)>;] end

Repeat

repeat <statement>
[<statement>; ...]end

Return

return [<expression> [, ...]]

Signal

signal <symbolic-expr>, <text-expr>

Wait

wait [for <interval> | until <logical-expr> checking every <interval> | until <event-predicate>]

Expressions

Using Generic References in Expressions

Throughout G2, you can generically reference sets of elements. The following quantifiers alert you to a generic reference: *the, any, every, each* and *a(n)*. The contexts in which you can use these quantifiers are:

the

Use: Can be used everywhere you use an expression.

Description: Specifies a unique value.

any

Use: Can only be used in the antecedent of rules.

Description: Specifies a generic reference in a rule.

every

Use: Can be used in two contexts: in *for every* expressions and in the consequent of rules.

Description: In a *for every* expression, it iterates over a set of elements. In the consequent of a rule, it causes an action to execute on every item that matches the reference.

each

Use: Can be used in two contexts: in arithmetic expressions over sets and for procedure iteration.

Description: In expressions over sets, it computes a value based on the set. In procedures, it iterates over items in a class.

a(n)

Use: Can only be used in *there exists* expressions.

Description: Specifies a single element to prove existence.

Animation

the item-height of <item>
the item-width of <item>
the item-x-position of <item>
the item-y-position of <item>
the icon-heading of <item>
the distance between <item> and (<item> | the nearest <item-class>)

Arithmetic

Operators: - (unary), +, -, *, /, ^
Formatting: as (<any dd.ddd format> | a time stamp | an interval)
EXPRESSIONS OVER SETS:
the (sum | product | average | minimum | maximum) over each <item-or-value-reference> [in (<g2-array> | <g2-list>)] of (<expression>)
the count of each <item-or-value-reference> [in (<g2-array> | <g2-list>)][such that (<logical-expression>)]

Array

(<g2-array> | <g2-list>) [<integer-expression>]
the array-length of <g2-array>
any <element type> in (<g2-array> | <g2-list>)
BY EXPRESSIONS OVER SETS: (See Arithmetic Expressions)

Attribute

the <attribute> of <item>
the <class> giving the <attribute> of <item>
the <class> that is an attribute of <item> (named by <symbolic-expression>)
<object> is <symbolic-value>
for any symbol that is a user-defined attribute of <item>

Connection

any <class> connected to <object>
any <class> connected at an (input to | input of | output of) <object>
any <class> connected at the (<portname> of | input to | input of | output of) <object>
any <class> at (an input end | an output end | either end) of <connection>

Existence

there exists (a | an) <item-or-value-reference> such that (<logical-expression>)
<item> exists
<item> does not exist

Hierarchy

CLASS:
the class of <item>
<item> is an instance of the class named by (<symbolic-expression>)
the symbol that is (a superior-class | an inferior-class) of <symbolic-expression>

Hierarchy continued

CLASS continued:
<item-or-value> is a <class-or-type>
WORKSPACE:
the <class> superior to <item>
the <class> upon <workspace>
the workspace of <item>
the subworkspace of <item>

History

the average value of <variable-or-parameter> (during the last <time-interval> | between <time-interval> ago and <time-interval> ago)
the integral in <time-unit> of <variable-or-parameter> (during the last <time-interval> | between <time-interval> ago and <time-interval> ago)
the interpolated value of <variable-or-parameter> as of <time-interval> ago
the maximum value of <variable-or-parameter> (during the last <time-interval> | between <time-interval> ago and <time-interval> ago)
the minimum value of <variable-or-parameter> (during the last <time-interval> | between <time-interval> ago and <time-interval> ago)
the number of history datapoints in <variable-or-parameter> (during the last <time-interval> | between <time-interval> ago and <time-interval> ago)
the rate of change per <time-unit> of <variable-or-parameter> (during the last <time-interval> | between <time-interval> ago and <time-interval> ago)
the standard deviation of <variable-or-parameter> (during the last <time-interval> | between <time-interval> ago and <time-interval> ago)
the value of <variable-or-parameter> (as of <time-interval> ago | as of <number> datapoints ago)

Item

BY NAME:
<item-name>
the <class> named by <symbolic-expression>
BY ATTRIBUTE:
the <attribute> of <item>
the <class> that is an attribute of <item> (named by <symbolic-expression>)
the <class> giving the <attribute> of <item>
BY CLASS HIERARCHY:
the class of <item>
<item> is an instance of the class named by (<symbolic-expression>)
<item-or-value> is a <class-or-type>
BY COMPARISON:
<item> is (not) the same object as <item>
BY CONNECTION: (See Connection expressions)
BY MEMBERSHIP:
the (first | second | next to last | last) <element-type> in the <list-type> list <list>
any <element-type> in (<g2-array> | <g2-list>)
(<item-array> | <item-list>) [<integer-expression>]
BY PROXIMITY:
the <class> nearest to <item>
BY RELATION: (See Relation expressions)
BY SCOPE:
(the item | this procedure | this procedure-invocation | this rule | this window | this workspace)
BY WORKSPACE HIERARCHY: (See Hierarchy expressions)

List

(<g2-array> | <g2-list>) [<integer-expression>]
the (first | second | next to last | last) <element-type> in the <list-type> list <list>
any <element-type> in (<g2-array> | <g2-list>)
<item-or-value> is (not) a member of the <list-type> list <list>
the number of elements in <list>
BY EXPRESSIONS OVER SETS: (See Arithmetic Expressions)

Logical

Logical operators: and, or, not
Relational operators: =, /=, is, is not, >, >=, <, <=
COMPARISON:
<arithmetic-expr> <relational-op> <arithmetic-expr>
<symbolic-expr> (= | /=) <symbolic-expr>
<symbolic-expr> (is | is not) <symbol>
<text-expr> <relational-op> <text-expr>
<item> is (not) the same object as <item>
FUZZY TRUTH:
Operators: is more true than, is less true than, is not more true than, is not less true than, =, /=
Fuzzy truth band: <quantity-expr> (< | <= | > | >= | =) <quantity-expr> (+- <quantity-expr>)
CONDITIONAL:
(if <logical-expr> then <expression> [else <expression>])
FOR EVERY:
for every <item-or-value-reference> (<logical-expr>)

Relation

any <class> that is <relation-to> <item>
<item1> is (not) <relation-to> <item2>
<item1> becomes <relation-to> <item2>
<item1> ceases to be <relation-to> <item2>

Symbolic

the name of <item>
the class of <item>
<symbolic-variable>
<symbolic-parameter>
the symbol <symbol>
the <symbolic-attribute> of <item>

Text

"text-message [<expression>]"
<text-variable>
<text-parameter>

Time

the current (time | real time | year | month | day of the month | day of the week | hour | minute | second)
the collection time of (<variable> | <parameter>) [as of <integer> datapoints ago]
the expiration time of <variable>

Value, Current Value, and Simulated Value

CURRENT VALUE:
the current value of (<variable> | (<expression>))
(<variable> | (<expression>)) has (a | no) current value
the first of the following expressions that has a current value (<list-of-expressions>)
VALUE:
the value of (<variable> | (<expression>))
(<variable> | (<expression>)) has (a | no) value
the first of the following expressions that has a value (<list-of-expressions>)
SIMULATED VALUE:
the simulated value of <variable-or-parameter>

Rules

For-Prefix Rule

for (any | the) <item> ...
(<if-rule> | <unconditionally-rule> | <when-rule> | <whenever-rule>)

If Rule

[for (any | the) <item> ...] if <logical-expression> then <action(s)>

Initially Rule

initially [if <logical-expression> then] <action(s)>

Unconditionally Rule

[for (any | the) <item> ...] unconditionally <action(s)>

When Rule

[for (any | the) <item> ...] when <logical-expression> then <action(s)>

Whenever Rule

[for (any | the) <item> ...]
whenever <event-preposition> [or <event-preposition>]
[and when <logical-expression>] then <action(s)>

LITERATUURLIJST

Advokaat, I.A.H., *Het gebruik van zetmeel in de papierindustrie; modellering voor planning* , Rijksuniversiteit Groningen, Faculteit der Economische Wetenschappen, Vakgroep Econometrie, Groningen, 1994.

Bergen, W.J.C. van e.a., *Het papierboek*, vierde druk, Stichting Vakopleiding Papierindustrie, Haarlem, 1986 (*non vidi*).

Brouwer, P.H., 'The interrelation between zetapotential and cationic or anionic demand to achieve highest retentions'. In: *1990 Papermakers Conference*, TAPPI, 1990.

Buchanan, B.G. en E.H. Shortliffe (editors), *Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Reading, Addison-Wesley, 1984 (*non vidi*).

Fieschi, M., *Artificial Intelligence in Medicine; Expert Systems* , Londen, 1990 [Oorspronkelijke uitgave: *Intelligence Artificielle en Médecine des Systèmes experts* , Parijs, 1984].

Gensym Corporation, *G2 Reference Manual version 3.0*, Gemsym Corporation, Cambridge, 1992.

Lucas, P.J.F. en L.C. van der Gaag, *Principes van expertsystemen*, Schoonhoven, 1988.

Luger, G.F. en W.A. Stubblefield, *Artificial Intelligence and the design of expert systems* , University of New Mexico, Albuquerque, 1989.

McGraw, K.L., *Designing and evaluating user interfaces for knowledge-based systems*, West Sussex, 1992.

Murdoch, S. en J. Johnson, *Intelligent data handling*, Londen etc., 1990.

Neal, J.G. en S.C. Shapiro, 'Intelligent Multi-Media Interface Technology'. In: Sullivan, J.W. (ed.), *Intelligent User Interfaces*, New York, 1991, blz. 11-43.

Newell, A., 'Production systems: models of control structures'. In: Chase, W.G. (ed.), *Visual information Processing*, Academic Press, New York, 1973.

Norman, D.A., *The psychology of everyday things*, New York, 1988.

Opdahl, P., *EPAK, an expert system for support of paper quality control*, z.pl, z.j.

Pearl, J., *Probabilistic Reasoning in Intelligent Systems.* Palo Alto, Morgan Kaufmann, 1988

Rich, E. en K. Knight, *Artificial Intelligence*, second edition, New York etc., 1991.

Roth, S.F., J. Mattis en X. Mesnard, 'Graphics and Natural Language as Components of Automatic Explanation'. In: Sullivan, J.W. (ed.), *Intelligent User Interfaces*, New York, 1991, blz. 207-239.

Smook, G.A., *Handbook for pulp and paper technologists*, second edition, Wilde Publications, Vancouver, 1992.

Stemerdink, J.D.B. e.a., *Mini KADS, een methode voor kleinschalige kennissysteemprojecten* , z.pl., z.j.

Verschoor, A.P.C., *Mini KADS, een studie naar de ontwikkeling van een kennissysteemontwikkelmethodologie voor kleinschalige projecten*, Universiteit van Amsterdam, Vakgroep Sociaal Wetenschappelijke Informatica, 1992.

Willemse, H.J. en G. Lindijer, *Software ergonomie*, Schoonhoven, 1988.

Yeager, R.L., 'Norske Skog uses expert system to optimize production quality'. In: "*Pulp and Paper*", december 1990.

INDEX

"

"vlag"-variabele 79

A

aardappelzetmeel 13
abstractie 19
actie 33
additief 5, 67
additieven 12
advies 5, 77
advieskennis 64, 66, 78
analyse 68, 75
anionic demand 17
antecedent 57
antischuimmiddel 12
antislijmmiddel 12
anti-slijmmiddel 13
approximate matching 37
automatisch redeneren 33
AVEBE 1, 3, 5

B

backward chaining 60
backward reasoning 33
backward-chaining 34, 59
bacteriegroei 12
Bayes theorie 39
Bayesiaans netwerk 42
beoordelingskennis 23
beslisbaar 31
beslisprocedure 31
best-first zoekmethode 38
bijtinten 12
bindmiddel 12
boomstructuur 73
breadth first 60

C

callback-procedure 74
cationic demand 17
cellulose 11
certainty factor 20
certainty factor model 40
classificatie 64
classificatiekennis 65
classificeren 19
coating 11
commando-interface 49
communicatie-ontwerp-model 7
conceptueel model 48
conditionele waarschijnlijkheid 39
conflict set 34
conflictresolutie-strategie 37
consequent 57
consultatie 33
control knowledge 60
control-knowledge 38
correctiekennis 23

D

data seeking 60
database 25
degree of substitution 17
demonstratiesysteem 20
Dempster-Shafer theorie 44
depth first 60
derivaat 14
diagnosekennis 78
direct manipulatión interface 52
doelgroep 20
doelstelling 5, 81
doelvariabele 34
domeinlaag 7, 61

E

E/R diagram 62
entiteiten 61
EPAK 22
event 34
expertisemodel 7, 61
expertsysteem 4, 27, 29

F

feedback 48, 81
feit 33
fixeermiddel 12, 13
focal-category 59
focal-class 59
focal-object 59
focusing 59
forward chaining 60

forward reasoning 34
forward-chaining 34, 59
functioneel ontwerp 9
fuzzy logic 26

G

G2 ..... 5, 23, 55
gebruikerstest 8
gegevensbeschrijving 33, 62, 75
gemodificeerd zetmeel 14
generieke regel 58
Gensym Corporation 55
globale gegevensverzameling 33
grafische user-interface (GUI) 52
GUIDE 54

H

heuristisch 33
houtpulp 11
hulpstoffen 12
hybride systeem 27

I

if regel 58
implementatie 73
indexering 35
inference engine 58
inference-definitie 77
inferentie 33
inferentie, data-gestuurd (bottom-up) 33
inferentie, doel-gestuurd (top-down) 33
inferentielaag 7, 61, 64
inferentie-mechanisme 58
inheritance 57
initialisatie 64
initialisatiekennis 65
initially regel 58
interval-object 65
interviews 61
invoking 59
inwendige sterkte 12

K

KADS 7
kationisch zetmeel 14, 16
kennis 75
kennisacquisitie 61
kennisdefinitie 75
kennis-editor 64, 82
kennisformalisme 64
kennisrepresentatie 29
kennisstructuur 61
kennisverzameling 77
kleurstof 12
knowledge base 33
kwaliteitsmatrix 23

L

lijmpers 11
lijmstof 12
load 68, 75
logica 29

M

mapping 48
matching 35
menu 51
menu-georiënteerde interface 51
metafoor 49
methode 57
Mini KADS 7
Mini-KADS methodiek 61
mode 56
modus ponens 31
multiple-choice menu 51
MYCIN 20, 41

N

natief zetmeel 13, 14
natpartij 11
natsterkte 12
natsterktemiddel 12
natuurlijke taal-interface 50
neuraal netwerk 26
non-determinisme van de eerste soort 35
non-determinisme van de tweede soort 35

O

objecthiërarchie 75
object-oriëntatie 57
onbeslisbaar 32
ontwatering 12
ontwateringsversneller 12
ontwerp 61
ontwikkelomgeving 5
onzekerheid 39
opaciteit 12
opdrachtomschrijving 3
oploopkast 11
oppervlaktelijming 12
oppervlaktesterkte 11
opwitter 12
ORACLE 25
organisatiemodel 7
overerving 57

P

papiereigenschappen 4, 12
papierfabricage 11
Papierlab 3, 5, 61
papiermachine 67
papierpulp 11
papiersterkte 12
papierwensen 4

pattern matching 23
pH 17
pH-regelaar 12
pilot project 8
planning 7
polymorfisme 57
predikaat 33
predikatenlogica 30
Probleemanalyse 25
probleemcontext 19
probleemdiagnose 64
probleemdiagnose-kennis 65
probleemstelling 3
proceseigenschappen 17
proceskennis 23
proceskoppeling 82
proces-omstandigheden 12
procesomstandigheden 13
proces-optimalisering 62
procesparameters 23
produktieregel 32, 57
produktiesysteem 32
projectdoelen 8
projectidentificatie 8
projectplanning 8, 10
projectspecificatie 8
projectvoorstel 8
ProMod 1
propositielogica 29
prototype 4, 5, 81
pulp samples 4
pulpmonster 67

R

regelinstantiatie (rule-instance) 38
regelverzameling 33, 78
relationeel datamodel 62
resolutie 32
responstijd 54
restart 75
restrictie 56
resultaat 81
RETE-match algoritme 36
retentie 12, 16, 17, 65
retentiemiddel 12, 16
retrogradatie 14

S

save 68, 75
scan-interval 59
scanning 59
semi-beslisbaar 31
sessie 33
SIX 1
specific conductivity 17
SPHINX 21
stoorstof 17
strategische laag 7
strijken 11
systeembouw 8
systeemspecificatie 8

T

taak-agent-model 7
taaklaag 7, 61
terugkoppeling 48
time-tag 38
tolerantie 48
traceren 34
tweezijdigheid 11

U

UIL 54
unconditionally regel 58
update-interval 60
user interface 45, 67, 74

V

variabele 33, 60
variabele, enkelwaardig (singlevalued) 33
variabele, meerwaardig (multivalued) 33
variabelen, vraagbaar (askable) 34
Vertis 1, 5
vierlagenmodel 7
vlag-variabele 66
vuistregel 32
vulstof 12

W

water-absorptie 12
wet-end 3, 11, 13, 65
when regel 58
whenever regel 58
workspace 56, 73

Z

zeef 11
zekerheidsfactor 20, 40
zetapotentiaal 17
zetmeel 13, 67
zetmeelderivaat 14
zetmeelstijfsel 12
zuurgraad 12