Het nut van kennissystemen

Het nut van kennissystemen

Robert V. Schuwer

HET NUT VAN KENNISSYSTEMEN

Het nut van kennissystemen

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van
de Rector Magnificus, prof. dr. J.H. van Lint, voor
een commissie aangewezen door het College
van Dekanen in het openbaar te verdedigen op
woensdag 12 mei 1993 om 16.00 uur

door

ROBERT VICTOR SCHUWER
Geboren te 's-Gravenhage

Dit proefschrift is goedgekeurd door de promotoren:
prof.dr. T.M.A. Bemelmans
en
prof.dr.ir. J.C. Wortmann

THESIS PUBLISHERS
AMSTERDAM 1993

Voor Sabine, Thomas en Wouter.

Omslagontwerp: Sabine Schuwer - van Heijst.
De afbeelding op de omslag is het symbool van het sterrenbeeld Tweelingen. In de astrologie geldt dit sterrenbeeld als het teken van kennis.

INHOUD

1 INLEIDING EN PROBLEEMSTELLING

1.1 INLEIDING

Kennissystemen zijn reeds vele jaren een van de verschijningsvormen van informatietechnologie. De vroegste en tegelijk meest bekende systemen ontstonden in de jaren 70. Veelal genoemd in de literatuur worden de systemen XCON voor de configuratie van VAX-computers (Harmon en Maus, 1988), MYCIN voor de diagnose van infectieziekten (Buchanan en Shortliffe, 1984) en PROSPECTOR voor de analyse van grondmonsters op de aanwezigheid van mineralen (Waterman, 1986). De projecten die geleid hebben tot het ontstaan van deze systemen kenmerken zich door een lange looptijd, een groot aantal betrokkenen en een prototype-achtige manier van ontwerpen. Tevens zijn de eerder genoemde systemen vaak van grote omvang en is gespecialiseerde hardware nodig om het systeem te gebruiken. Het systeem XCON bijvoorbeeld bevat momenteel zo'n 10.000 kennisregels. Het systeem MYCIN is geprogrammeerd in Lisp en draait op een Lisp-machine.

In de jaren '80 ontstond, mede door de opkomst van de PC, een golf van publicaties over kennissysteemprojecten. Meestal betrof het systemen van een relatief kleine omvang (in vergelijking met de hiervoor genoemde systemen), die stand-alone draaiden op een PC en die werden gerealiseerd met behulp van specifieke software (zgn. kennissysteemshells). Veelal waren de probleemtypen waarvoor een kennissysteem werd ontwikkeld, van het type diagnose of classificatie. Er is echter een trend zichtbaar waarbij de aandacht verschuift naar plannings- en schedulingproblemen, die met behulp van een kennissysteem worden aangepakt. Op het in 1991 in de VS gehouden World Congress on Expert Systems werd gerapporteerd over een 160-tal applicaties, waarbij meer dan de helft een schedulingprobleem betrof (Liebowitz, 1991).

Kennissystemen worden veelal ontwikkeld via de rapid-prototyping aanpak. Bij deze aanpak wordt eerst een deel van de kennis achterhaald, waarna dit geprogrammeerd wordt in een prototype kennissysteem. Met behulp van dat prototype wordt de rest van de kennis achterhaald en de reeds ingebrachte kennis gevalideerd. Het uiteindelijke systeem ontstaat zo op een incrementele wijze vanuit het initiële prototype. De laatste jaren is er veel aandacht voor een meer methodische aanpak van het ontwikkelproces, naar analogie van de traditionele, meestal gefaseerde software ontwikkeling. Voorbeelden van methoden in dit kader zijn KADS (Breuker et al, 1987) en ESDM (Kuiper, 1988). Vaak wordt bij deze methoden het ontwikkelen van een prototype voorgeschreven om meer inzicht te verkrijgen in bijvoorbeeld de haalbaarheid ervan.

De rapid-prototyping aanpak werd mede mogelijk gemaakt door de komst van zgn. *shells*. Een shell is het beste te omschrijven met de term: leeg kennissysteem. In zijn meest eenvoudige vorm bestaat een shell uit een standaard redeneermechanisme, een gebruikersinterface en hulpmiddelen voor het inbrengen van de kennis. Iets geavanceerdere shells bieden mogelijkheden zelf het redeneermechanisme en de gebruikersinterface aan te passen. Over het algemeen wordt het met behulp van een shell mogelijk op een snelle wijze een kennissysteem te programmeren waarbij vaak veel aandacht is besteed aan het aspect gebruiksvriendelijkheid.

1.2 PROBLEEMSTELLING

Hoewel het aantal projecten waarin een kennissysteem werd ontwikkeld in de jaren 80 sterk groeide, werd opvallend weinig gerapporteerd over *succesvolle* applicaties. Hierbij wordt onder een succesvolle applicatie een kennissysteem verstaan, dat daadwerkelijk uitontwikkeld is en wordt gebruikt in een operationele omgeving. Een onderzoek in Groot Brittannië, dat in 1990 verscheen en waarbij meer dan 100 kennissysteemprojecten waren betrokken, wees uit, dat de meeste van de ontwikkelde systemen "slechts" prototypes waren (Liebowitz, 1991, pp 1950-1958). Er zijn diverse oorzaken aan te voeren voor dit verschijnsel.

1. Kennissystemen ontstonden in een bedrijfsmatige omgeving vaak nadat in een researchomgeving ervaring met de nieuwe technologie was opgedaan (Broek et al, 1990). Doordat in de researchomgeving de gebruiker van het systeem en de ontwikkelaar vaak dezelfde persoon was, legde dit weinig eisen op aan de gebruikte methode van ontwikkelen en de onderhoudbaarheid en inzichtelijkheid van het resulterende systeem. De meeste systemen in een researchomgeving werden daarom ook ontwikkeld met behulp van de prototyping aanpak. Door het ontwikkelen met een prototyping aanpak ontstaat echter een systeem, waarin de kennis veelal ongestructureerd aanwezig is (Broek et al, 1990), (Breuker et al, 1987). Het "klakkeloos" overnemen van deze ontwikkelaanpak in een bedrijfsmatige omgeving levert problemen op, die vooral liggen op het gebied van beheersbaarheid van het ontwikkelproces en de onderhoudbaarheid van het ontwikkelde systeem door de ongestructureerdheid van de kennis. In (Breuker et al, 1987) wordt dit genoemd als een aanleiding voor het meer methodisch ontwikkelen van een kennissysteem. Daarbij ontstaat een model van de kennis voordat met de implementatie wordt begonnen. Doordat zo'n model ontbreekt bij systemen die met een prototyping aanpak zijn ontwikkeld, zijn dergelijke systemen minder inzichtelijk en daardoor slecht

onderhoudbaar. Een zodanig prototype geeft dan aanleiding tot de beslissing niet verder te gaan met de ontwikkeling ervan tot een volwaardig systeem.

2. Het fenomeen "kennissysteem" is nog steeds omgeven met een waas van geheimzinnigheid. In gesprekken met ontwikkelaars van kennissystemen bij adviesbureaus en softwarehuizen bleek, dat veel meer kennissystemen ontwikkeld worden en daadwerkelijk in een operationele omgeving worden gebruikt dan uiteindelijk in publicaties is af te lezen. Veelal werd als reden aangegeven, dat de meeste van deze systemen zodanige kennis bevatten, dat de inhoud, maar ook het gebruik ervan, uit concurrentie-overwegingen geheim moest blijven.
3. Veel kennissystemen maken gebruik van gegevens, die reeds zijn vastgelegd in bestaande databases binnen een organisatie. Door het stand-alone karakter van het (prototype) kennissysteem is het echter vaak niet mogelijk, deze gegevens te benaderen vanuit het kennissysteem. Zeker indien een kennissysteem veel gegevens nodig heeft voor het uitvoeren van de taak, is het inefficiënt deze gegevens opnieuw in te brengen in plaats van rechtstreeks in te lezen vanuit een database.

Punt 3 geeft aan, dat het vraagstuk van integratie van een kennissysteem in een reeds bestaande, conventionele informatiesysteemstruktuur een belangrijke succesfactor is voor het al dan niet slagen van een kennissysteemproject (Liebowitz, 1991). Hier sluit de volgende constatering bij aan. Uit de beschrijvingen van applicaties kan worden geconstateerd, dat bij de ontwikkeling er reeds vanuit wordt gegaan, dat inderdaad een kennissysteem moet worden ontwikkeld. Er wordt echter in de literatuur geen aandacht besteed aan de situatie, waarbij in een conventioneel systeemontwikkeltraject delen van het systeem mogelijk een kennissysteemoplossing vereisen. Deze herkenning van dergelijke mogelijkheden bij de ontwikkeling wordt bemoeilijkt door tenminste twee factoren:

1. Er bestaat geen eensluidende opvatting over het begrip "kennissysteem". De meeste in de literatuur gegeven definities blijven vaag en geven onvoldoende inzicht in het specifieke van een kennissysteem. Een gevolg is dat ook niet kan worden afgeleid wanneer een kennissysteem een goede wijze van oplossen van een probleem is en welke voordelen dit biedt ten opzichte van alternatieve oplossingen.
2. Het is onduidelijk op welk moment in een ontwikkeltraject deze herkenning moet plaatsvinden en welke activiteiten er extra moeten worden ondernomen om zo'n ontwikkeling mogelijk te maken.

Deze studie zal ingaan op de hier genoemde problemen. De voorgaande opsomming van problemen leidt tot de volgende *probleemstelling*

Het is onduidelijk in welke situaties een kennissysteem een goede oplossing is voor een probleem en welke toegevoegde waarde deze oplossing dan heeft ten opzichte van alternatieven.

Een eerste analyse van de probleemstelling leert, dat allereerst het begrip "kennissysteem" zal moeten worden gedefinieerd. De definitie zal zodanig moeten zijn, dat hieruit uitspraken kunnen worden afgeleid over toepassingsmogelijkheden voor kennissystemen en over de toegevoegde waarde die een kennissysteem dan heeft. Vanuit de probleemstelling en deze constatering kan daarom de volgende *hoofddoelstelling* van deze studie worden geformuleerd:

1. Geef een eenduidige definitie van een kennissysteem.
2. Geef, uitgaande van de definitie van een kennissysteem ontwerprichtlijnen die tot een verantwoorde keuze leiden wanneer een kennissysteem een goede oplossing is voor bepaalde probleemsituaties.

Tijdens het uitvoeren van de studie bleek, dat de ontwikkelde definitie van een kennissysteem tevens gebruikt kon worden als kader voor het opstellen van een *conceptuele specificatie* van een kennissysteem tijdens de fase "logisch ontwerp" in een ontwikkeltraject. Tevens kan de definitie gebruikt worden voor het op bepaalde punten onderling *vergelijken van software hulpmiddelen* (programmeertalen of shells).

De rest van het boek heeft de volgende inhoud. In hoofdstuk 2 wordt antwoord gegeven op de vraag "*WAT* is een kennissysteem". Het geeft een inventarisatie en classificatie van definities van kennissystemen. Tevens wordt op een informele wijze de in deze studie ontwikkelde definitie beschreven. Ter illustratie wordt een beschrijving gegeven van een bestaand kennissysteem (MYCIN) in de terminologie van de ontwikkelde definitie.

In hoofdstuk 3 wordt antwoord gegeven op de vraag "*HOE* ziet een specificatie van een kennissysteem eruit". Hierin wordt een beschouwing gegeven over hoe de ontwikkelde definitie kan worden gebruikt bij het maken van een conceptuele specificatie van een kennissysteem. Deze wijze van specificeren wordt vergeleken met twee andere wijzen van specificeren uit respectievelijk de methoden KADS en DESIRE. In hoofdstuk 4 wordt ter illustratie van het specificeren van een kennissysteem volgens de richtlijnen uit hoofdstuk 3 een voorbeeld van een systeem uitgewerkt. Het betreft een systeem, dat bij het configureren van een complex produkt (een bureau-eiland) controleert of een opgegeven configuratie voldoet aan alle voorwaarden, die aan een eindprodukt zijn opgelegd. Tevens worden twee implementaties besproken van dit systeem. De implementaties dienen als illustratie hoe de ontwikkelde definitie uit hoofdstuk 2 kan worden gebruikt voor het vergelijken van

verschillende software hulpmiddelen. De hoofdstukken 3 en 4 beschrijven als zodanig een nevenresultaat van de studie.

In hoofdstuk 5 wordt aangegeven *WANNEER* een kennissysteem geschikt is om als oplossing te gebruiken in een specifieke situatie. Hierbij worden richtlijnen aangegeven, waarmee kan worden vastgesteld in welke situaties een kennissysteem een geschikte wijze van oplossen biedt. Dit hoofdstuk vormt samen met hoofdstuk 2 de kern van de studie.

Een van de resultaten uit hoofdstuk 5 is dat integratie van een kennissysteem een kritieke succesfactor is voor het daadwerkelijk gebruiken ervan. In hoofdstuk 6 wordt stilgestaan bij aspecten van integratie van kennissystemen. Aan de orde komen hierbij onder meer welke vormen van integratie te onderscheiden zijn. Er wordt tevens verslag gedaan van een praktijkonderzoek naar daadwerkelijk gerealiseerde, geïntegreerde kennissystemen.

Hoofdstuk 7 bevat de conclusies van deze studie en aanbevelingen voor verder onderzoek. Het boek wordt afgesloten met twee bijlagen. In bijlage A wordt de ontwikkelde definitie op een formele, wiskundige wijze gepresenteerd. Bijlage B bevat een formele specificatie van de database voor de configuratie van het bureau-eiland (het voorbeeld uit hoofdstuk 4).

2 DEFINITIES VAN EEN KENNISSYSTEEM

In dit hoofdstuk zullen verschillende definities voor een kennissysteem vanuit de literatuur worden gepresenteerd. Het doel is om te komen tot een ordening van definities en tot een eerste afbakening van het onderwerp van dit proefschrift. Aan het einde van dit hoofdstuk zal een keuze worden gemaakt betreffende de te volgen terminologie in dit proefschrift.

In de literatuur worden de termen kennissysteem en expertsysteem door elkaar gebruikt. Als er al onderscheid wordt gemaakt, dan is het onderscheidende aspect de bron van de kennis en/of de prestaties van het systeem. Bij een expertsysteem is een menselijke expert de bron van kennis en presteert het systeem vergelijkbaar met de expert. Bij een kennissysteem wordt de kennis niet noodzakelijkerwijs verkregen van een menselijke expert en hoeft het systeem ook niet net zo goed als een menselijke expert te presteren.

2.1 INTELLIGENTIE-DEFINITIES

Definities van kennis- en expertsystemen sluiten nauw aan bij de verschillende AI-onderzoeksinteressen. Een eerste ordening van definities wordt daarom verkregen door uit te gaan van AI-onderzoek.

Een deel van AI-onderzoek richt zich op de wijze waarop denkprocessen bij de mens zich afspelen. Dit AI-onderzoek zou inzicht moeten verschaffen in de wijze waarop een mens denkt en redeneert. Tevens zou AI-onderzoek dan "intelligent" gedrag van een computer moeten opleveren. Intelligentie wordt hierbij beschreven als "vergelijkbaar met het menselijk denken en redeneren".

Definities van kennis- en expertsystemen, die aansluiten bij dit type onderzoek, benadrukken het "intelligente" gedrag van een kennissysteem als karakteriserend. Intelligentie zou zich dan moeten uiten in de taken, die een expertsysteem uitvoert. Zo definiëren Weitz en De Meyer (1990) een expertsysteem als:

> "An expert system is a computer program for solving difficult, "fuzzy" problems in domains where human expertice is normally associated with a great deal of training and experience." (Weitz en De Meyer, 1990, blz. 116)

Madni (1988) benadrukt in zijn definitie daarnaast de mogelijkheid van verklaring van redeneer- en keuzegedrag van een expertsysteem:

> "Expert systems are computer programs that attempt to emulate the problem-solving and decision-making performances of experts. (..) The goal of expert systems is to solve specific problems and present and "explain" the solution in terms comprehensible to humans." (Madni, 1988, blz. 395)

Hayes-Roth, Waterman en Lenat (1983) omschrijven een expertsysteem met behulp van een zevental karakteristieken. Onder meer wordt expertise daarbij beschreven als het gebruiken van declaratieve kennis en het vermijden van "blind" zoeken. Tevens worden de type taken, die met behulp van een expertsysteem worden opgelost, beschreven in generieke termen: interpretatie, diagnose, voorspelling, instructie, monitoring, planning en design. Deze laatste taakindeling is nader uitgewerkt ook te vinden in (Breuker et al, 1987).

Szolovits (1989) geeft in zijn definitie aan dat een kennissysteem niet per se gebruik hoeft te maken van nieuwe technologieën:

> "Knowledge-Based Systems are computer programs, *usually* based on technologies developed by AI research, which embody some aspects of human knowledge and expertise to perform tasks ordinarily done by human experts." (Szolovitz, 1989, blz. 339)

Soortgelijke definities als hiervoor zijn te vinden in (Jackson, 1987), (Weiss en Kulikowski, 1984).

Waterman (1986) definieert een expert systeem als:

> "A computer program using expert knowledge to attain high levels of performance in a narrow problem area." (Waterman, 1986, blz. 11)

Als kenmerkend voor expert knowledge ziet hij het gebruik van heuristieken (vuistregels die de oplossingsruimte effectief verkleinen bij het zoeken naar een oplossing) en het gebruik van onzekerheid als een maat van vertrouwen voor de validiteit van een feit of regel.

2.2 ARCHITECTUURDEFINITIES

Kennissystemen kunnen ook worden benaderd vanuit de historische ontwikkeling van automatisering. Zoals reeds werd beschreven in (Schuwer en Kusters, 1990), kenmerkt de automatisering zich vanaf het begin, door een steeds verder gaande modularisatie van met name software. Men kan m.b.t. software het volgende onderscheid maken:

- operating systeem en utilities (systeemsoftware)
- software voor data-opslag en -retrieval
- applicatieprogrammatuur

Bij de allereerste computerprogramma's waren alle drie de componenten terug te vinden in één programma. Er was geen sprake van een apart operating systeem. Elk programma bevatte eigen laad-, lees- en printfuncties.

De eerste hoofdscheiding in componenten betrof een scheiding tussen systeemsoftware en bewerkingssoftware. Doordat men niet bij ieder programma ook de systeemtaken hoefde te programmeren, werd een grote efficiency-winst bereikt. Hoewel de systeemsoftware aanvankelijk slechts de eerder genoemde laad-, lees- en printfuncties bevatte, leidde het ontstaan van multiprogramming later tot meer complexe operating systemen vanwege de noodzaak van intern geheugenbeheer.

De tweede hoofdscheiding in componenten betrof de scheiding tussen data en applicatieprogrammatuur. Toen daarna tevens de mogelijkheid van gelijktijdig gebruik van gegevens door meer gebruikers ontstond, leidde dit tot het afsplitsen van de gegevensbeheersfuncties uit de applicatieprogrammatuur. Zo ontstonden Database Management Systemen (DBMS).

Een volgende stap in deze modularisatietrend betrof de afscheiding van kennis over de gegevens uit de applicatieprogrammatuur. Deze stap werd genomen bij de eerste "klassieke" expertsystemen in de betekenis, zoals hiervoor geschetst. Zo ontstond tevens het besef van explicitering van de kennis over de data. Deze kennis is bij veel hedendaagse toepassingen nog in de applicatieprogrammatuur terug te vinden. In feite wordt zo één type kennis onderscheiden.

Definities van kennissystemen, die gebaseerd zijn op voorgaande ontwikkelingen, benadrukken vooral de architectuur van zo'n type systeem. Tevens wordt vaak het (beperkte)

domein, waarop een kennissysteem werkzaam is, in de definitie genoemd. Zo definieert Mars (1988) een kennissysteem als:

> "Een *kennissysteem* is een computerprogramma, waarin zo goed mogelijk een scheiding is aangebracht tussen toepassingsgebiedonafhankelijke afleidingsregels en toepassingsgebied-specifieke kennis." (Mars, 1988, blz. 86)

Voorbeelden van toepassingsgebiedonafhankelijke afleidingsregels zijn modus ponens en resolutie. Mars onderscheidt tevens het begrip *strategische kennis*. Strategische kennis is kennis over kennis (metakennis). Het geeft aan in welke situaties welke domeinkennis moet worden toegepast. Het toevoegen van strategische kennis heeft tot doel het zoeken naar een oplossing door het systeem te versnellen. Zo heeft een roosterplanner veel kennis, die gebruikt wordt om *snel* een zo goed mogelijk rooster te genereren. Deze kennis geeft geen informatie over de *inhoud* van een rooster, maar over het *proces*, waarmee zo efficiënt mogelijk een rooster wordt gegenereerd. Strategische kennis kan deels toepassingsgebied-specifiek zijn en deels toepassingsgebied-onafhankelijk.

Mars maakt tevens een expliciet onderscheid tussen een kennissysteem en een expertsysteem, getuige zijn definitie van een expertsysteem:

> "Een *expertsysteem* is een kennissysteem dat op zijn gebied tenminste even goede prestaties levert als een menselijke expert." (Mars, 1988, blz. 86)

Ook Waterman (1986) maakt een onderscheid tussen expertsysteem en kennissysteem. Naast zijn definitie van een expertsysteem, die in paragraaf 2.1 werd gegeven, hanteert hij de volgende definitie van een kennissysteem:

> "A knowledge based system is a program in which the domain knowledge is explicit and separate from the program's other knowledge." (Waterman, 1986, blz. 23)

De Jong (1988) geeft in zijn definitie aan, dat de kennis die in een kennis- of expertsysteem wordt gebruikt te vinden is in een zgn. kennisbank:

> "Expert systems are software programs for emulating the problem solving behaviour of humans. The knowledge of a human expert is stored in a knowledge base. Knowledge consists generally of problem knowledge (domain knowledge) and knowledge how to solve the problem (meta knowledge)" (De Jong, 1988, blz. 418)

Hij gaat tevens in op verschillen tussen een expertsysteem en een traditioneel systeem:

> "There are important differences between an expert system and conventional software:
>
> * Contrary to the database, the knowledge base of an expert system also contains information on how the data should be handled.
> * Contrary to a traditional software program, the inference engine of an expert system contains no information on the problem domain.
> * All knowledge in an expert system is represented in a declarative way. A declarative representation is referentially transparent: the meaning of the knowledge as a whole is fully determined by the meaning of the individual knowledge parts and not from e.g. a sequential order in which the parts appear on the knowledge base." (De Jong, 1988, blz. 418)

Evenals bij Mars wordt hier dus onderscheid gemaakt in kennis over het probleemgebied en kennis over het redeneerproces, die met behulp van de eerste soort kennis af te leiden is. Deze laatste soort kennis wordt verder in dit boek aangeduid met *inferentie*, *redeneerkennis* of *metakennis*.

Ook Frost (1987) gaat in zijn definitie van een kennissysteem nader in op het verschil met een conventioneel systeem. Hij noemt onder andere de automatische controle op integriteit van de data bij gebruik van een kennissysteem:

> "A knowledge base system is a set of resources - hardware, software and possibly human - whose collective responsibilities include storing the knowledge base, maintaining security and integrity, and providing users with the required input/output routines, including deductive retrieval facilities, so that the knowledge base can be accessed as required. Knowledge base systems,

as currently discussed in the literature, are distinct from conventional database systems in four ways:

a. Knowledge bases typically contain explicitly represented rules as well as simple facts
b. Knowledge base storage structures typically have low structural semantic content compared with database structures.
c. Knowledge base systems include components for the automatic maintenance of semantic integrity in addition to components for syntactic checking as found in conventional database systems.
d. Knowledge base systems include components which can make inferences over the knowledge base, thereby providing a deductive retrieval facility." (Frost, 1987, blz. 5)

Buchanan en Shortliffe (1984) hebben in hun definitie zowel aspecten van intelligentie-onderzoek als architectuur:

"An expert system is an AI program designed

(a) to provide expert-level solutions to complex problems,
(b) to be understandable, and
(c) to be flexible enough to accomodate new knowledge easily." (Buchanan en Shortliffe, 1984, blz. 3)

2.3 VERGELIJKING

Hoewel de laatste jaren talloze boeken en artikelen verschenen zijn met kennis- en expertsystemen als onderwerp, is het opvallend, dat de meeste auteurs niet omschrijven wat ze onder zo'n type systeem verstaan. Dit zou er op kunnen wijzen dat de term reeds een betekenis heeft, die eenduidig en ingeburgerd is (zoals dit voor bijvoorbeeld het begrip 'database' geldt). De verschillende definities en hun onderlinge verschillen hebben echter aangetoond, dat dit laatste geenszins het geval is. Hierdoor blijft het vaak onduidelijk, welk beeld van een kennissysteem de auteurs in gedachten hadden bij hun beschrijving van het systeem. Veel resultaten die in zulke literatuur worden beschreven, blijven juist daardoor vaag. Zijn de resultaten te verklaren doordat een systeem met een architectuur van een

kennisbank werd ontwikkeld, of doordat gebruik werd gemaakt van de ervaringen van een expert op het onderhavige terrein?

Bij de meeste genoemde definities uit paragraaf 2.1 en 2.2 kan voor een systeem niet eenduidig worden vastgesteld, of het volgens zo'n definitie inderdaad een kennissysteem is. Als een globale toets voor de exactheid van een definitie kan de "Boekhoudtest" worden gebruikt:

> Een boekhouder is een persoon, wiens beroep het is voor een koopman, een instelling, een maatschappij etc. boek te houden, dwz. volgens bepaalde regels in daartoe ingerichte boeken schrijven wat er in een handel of huishouding plaats heeft. Een van de taken, die een boekhouder moet uitvoeren, is het journaliseren van posten. Een boekhouder kan bij het uitvoeren van die taak worden ondersteund door geautomatiseerde systemen. Omdat de regels van het boekhouden nogal complex zijn, kan een ervaren boekhouder zeker een expert worden genoemd. Het ondersteunende systeem voor het journaliseren van posten wordt echter nooit beschouwd als een kennis- of expertsysteem. Ervan uitgaande dat dat correct is, kan een definitie van een kennis- of expertsysteem, die ook zo'n boekhoudprogramma aanmerkt als zijnde zo'n type systeem, als onvoldoende scherp worden beschouwd.

Als de "boekhoudtest" wordt toegepast op de eerder behandelde definities, dan is de definitie van Szolovits zonder meer onvoldoende. De definitie van Weits en De Meyer is, op het woord "fuzzy" na, ook van toepassing op een boekhoudprogramma. De definitie van Madni kan slechts deels worden toegepast op een boekhoudprogramma. Het verklaren van een oplossing en het nabootsen van probleemoplos- en beslisgedrag zijn aspecten, die in een boekhoudprogramma niet voorkomen. Ook de omschrijving van Hayes-Roth c.s. voor een expertsysteem meerdere elementen, die niet van toepassing zijn op een boekhoudprogramma.

Het is opvallend, dat de definities van de tweede soort (Mars, De Jong, Frost en deels Buchanan en Shortliffe (aspect (c))) alle de "boekhoudtest" doorstaan. Dit is te verklaren door de aandacht die in deze definities wordt geschonken aan de architectuur van het systeem in plaats van de soort kennis, die in het systeem opgeslagen is.

Iedere definitie heeft een waarde in een bepaalde context. Zo zullen de definities, die uitgaan van het vastleggen van expertkennis op een zodanige wijze, dat het systeem presteert als een menselijk expert, vooral interessant zijn voor cognitief psychologen. Voor hen is de architectuur van zo'n systeem minder belangrijk. Tevens is zo'n definitie van belang voor

praktijkmensen, die voor bepaalde taken een systeem willen bouwen ter ondersteuning. In (Mars, 1991) wordt dit de *processchool* genoemd. Dit is in tegenstelling met de aanhangers van de *produktschool*, waarbij het uiteindelijke produkt van belang is en waar in het produkt niet perse de werkwijze van een expert hoeft te zijn gemodelleerd.

In dit proefschrift zal worden uitgegaan van een architectuur-definitie van een kennissysteem. De reden om dit te doen is dat daarbij wordt aangesloten bij de modularisatie-trend. Door verder een database expliciet in de architectuur van een kennissysteem op te nemen, wordt het belang van een database voor een kennissysteem benadrukt. Een andere reden voor deze keuze is, dat een kennissysteem wordt beschouwd als een bijzonder type informatiesysteem. Vooral van belang is dan de wijze, waarop een kennissysteem kan worden gekarakteriseerd zodanig, dat het verschil met andere typen informatiesystemen duidelijk wordt. In deze context moet ook de eerder beschreven "boekhoudtest" worden gezien: definities, die ook op boekhoudprogramma's van toepassing zijn, geven niet voldoende het onderscheid weer tussen een kennissysteem en een ander type informatiesysteem.

Sterk gerelateerd aan kennissystemen zijn de begrippen logic program en deductive database. De idee achter een logic program is, dat logica (meestal eerste orde predicatenlogica) gebruikt kan worden als een programmeertaal. Als belangrijkste resultaat van onderzoek in die richting kan de taal Prolog worden genoemd. Lloyd (1987) omschrijft een deductive database als een verzameling uitspraken, gedaan mbv. een logische programmeertaal, van een zekere vorm, waarover queries gesteld kunnen worden. Door middel van afleidingsregels is de deductive database in staat om een query te beantwoorden. In de rest van dit proefschrift zal een deductive database beschouwd worden als een bijzonder soort kennissysteem, namelijk een die m.b.v. een logische programmeertaal is geschreven.

Geen van de definities, die in de literatuur zijn gevonden, zijn exact genoeg om de kern van een kennissysteem precies te omschrijven. Dit geldt zonder meer voor de definities van de eerste soort. Ook de definities van de tweede soort zijn, hoewel ze meer houvast geven, niet goed genoeg. Zo is bij de definitie van Mars het onderscheid tussen toepassingsgebied-specifiek en toepassingsgebied-onafhankelijk te vaag om te dienen als een solide basis, omdat het begrip "toepassing" in sommige gevallen niet scherp omlijnd kan worden. Dit heeft als nadeel, dat niet over kennissystemen kan worden geredeneerd. Hierdoor is het moeilijk om gefundeerde uitspraken te doen over toepasbaarheid van kennissystemen en de toegevoegde waarde ervan.

Om deze redenen zal een exacte definitie van een kennissysteem worden opgebouwd. Dit zal een formele definitie zijn vanwege het voordeel van de eenduidigheid van de gebruikte taal

en de mogelijkheden om eigenschappen af te leiden uit gegeven definities. Als taal zal de eerste orde logica worden gebruikt. Deze taal biedt voldoende zeggingskracht voor dit doel. Tevens is deze taal relatief makkelijk verstaanbaar.

Deze definitie zal zoveel als mogelijk aansluiten bij de eerder gegeven informele architectuur-definities. De architectuur zal echter meer in detail worden gedefinieerd. Tevens zullen de relaties tussen de diverse onderscheiden componenten worden aangegeven. Door het op detailniveau exact vastleggen van de architectuur van een kennissysteem is het ook duidelijker, welke ontwerpstappen genomen moeten worden bij het ontwikkelen van een kennissysteem. Tevens kan meer gefundeerd een keuze gedaan worden uit beschikbare software-hulpmiddelen, die voor de realisatie van een kennissysteem kunnen worden gebruikt. In hoofdstuk 4 zal dit laatste nader worden uitgewerkt.

2.4 EEN DEFINITIE VAN EEN KENNISSYSTEEM

In deze paragraaf zal een definitie van een kennissysteem worden opgebouwd. Hierbij zal worden uitgegaan van de resultaten uit (Eiben en Schuwer, 1991) en (Lloyd, 1987). De in deze paragraaf te geven beschrijving is een informele weergave van het formeel wiskundige model, dat te vinden is in bijlage A. In de tekst zal, waar nodig, aan deze bijlage worden gerefereerd.

De opbouw van deze paragraaf is als volgt. Allereerst zullen in paragraaf 2.4.1 enkele beperkingen worden aangegeven van de definitie. Vervolgens wordt in paragraaf 2.4.2 de definitie gegeven.

De definitie zal worden geïllustreerd met het volgende voorbeeld:

> In het Nederlandse Koningshuis zijn diverse familierelaties aan te geven. Neem aan dat er behoefte is aan een systeem dat vragen moet kunnen beantwoorden van de vorm 'Geef de namen van alle kleinkinderen van Prins Bernhard' of 'Is Willem-Alexander een broer van Johan-Friso?' Zo'n systeem zal enerzijds generieke afleidingsregels bezitten, zoals de regel "Als persoon X de vader is van persoon Y, dan is persoon Y een kind van persoon X". Anderzijds zal zo'n systeem ook specifieke regels bevatten, die voor deze speciale familie gebruikt kunnen worden. Een voorbeeld is de redeneerstrategie

die stelt, dat bij vragen over de huidige generatie koningskinderen beter eerst naar zonen dan naar dochters kan worden gezocht.

2.4.1 BEPERKINGEN

De definitie van een kennissysteem zal zich beperken tot drie belangrijke componenten van een kennissysteem. Deze componenten zijn een database, een regelbank en een redeneermechanisme. De definitie zal ieder van die componenten definiëren en tevens de verbanden aangeven tussen die componenten.

Onzekerheid in de data, de regels of het redeneermechanisme zal aanvankelijk niet worden verondersteld. In werkelijkheid zal kennis vaak wel onzekerheid bevatten. Een kennissysteem splitst in zo'n geval het redeneerproces in twee delen: een 'echte' redenering, waarbij door het systeem wordt geprobeerd feiten af te leiden, en een (algoritmische) berekening, waarbij de zekerheid wordt berekend, waarmee het kennissysteem de uit de redenering gevonden conclusie kan presenteren. In paragraaf 2.5 zal worden getoond, dat het weglaten van onzekerheid geen ernstige beperking is van de formele definitie.

De definitie zal geen aandacht schenken aan gebruikers-interface-eisen. In het bijzonder zal geen aandacht worden besteed aan de wijze, waarop een systeem een uitleg geeft van de resultaten, die het heeft bereikt. In de praktijk is dit overigens vaak beperkt tot het presenteren van de regels, die gebruikt zijn om tot een conclusie te komen (de HOW-optie: Hoe is het systeem aan een conclusie gekomen) c.q. die het systeem wil gebruiken tijdens een redenering (de WHY-optie: Waarom moet de gebruiker een bepaalde vraag beantwoorden). In de volgende paragraaf zal wel worden aangegeven, waar in het op te bouwen model van een kennissysteem een uitlegcomponent kan worden ingepast. In hoofdstuk 5 zal het geven van uitleg als functionaliteit van een (kennis)systeem nader worden bekeken.

Tenslotte zal de definitie het gebruik van onvolledige informatie niet toestaan. Wat in dit kader onder "onvolledige informatie" moet worden verstaan, zal verderop worden behandeld.

2.4.2 DEFINITIE

In deze paragraaf zal uiteindelijk een architectuurmodel van een kennissysteem worden gepresenteerd. Dit model wordt in de rest van dit boek als definitie van een kennissysteem

gebruikt. Behalve de architectuur van het kennissysteem zal ook eenduidig worden aangegeven, welke taken ieder van de onderscheiden componenten van een kennissysteem heeft bij het uitvoeren van zijn, eveneens eenduidig omschreven, taak.

Om tot het model te komen zal worden uitgegaan van een eerste, voorlopige omschrijving:

> Een kennissysteem is een computerprogramma dat in eindige tijd een vraag beantwoordt op basis van
>
> - een gegeven verzameling regels,
> - een gegeven database en
> - alle gegevens die met behulp van de regels vanuit de database afgeleid c.q. berekend moeten worden om de vraag te beantwoorden.

De zinsnede "op basis van" uit deze omschrijving geeft weer dat het antwoord, dat door het kennissysteem gegenereerd wordt, een causaal verband heeft met de gegevens, die in de database aanwezig zijn of die berekend zijn. De zinsnede "in eindige tijd" sluit uit, dat een systeem in theorie tot een antwoord kan komen, maar daarvoor oneindig lang gegevens moet berekenen. In (Lloyd, 1987) staat een voorbeeld van een kennissysteem, waarbij het beantwoorden van een vraag oneindig lang duurt. Dit type kennissystemen zal in de rest van dit boek niet worden beschouwd.

Welke databases wel en welke niet toegelaten zijn, staat beschreven in een database-definitie. Het systeem is in staat om van iedere database na te gaan, of die al dan niet toegelaten is. Het gebruikt hiervoor een niet nader omschreven *controle mechanisme*. Dit "database"-deel van een kennissysteem is in feite niet meer dan een database-systeem.

Een regel is een uitdrukking van de vorm:

> ALS een aantal gegevens een waarheidswaarde TRUE hebben
> DAN geldt, dat de waarheidswaarde van een (afgeleid) gegeven TRUE is

Het ALS-deel van een regel wordt de staart van de regel genoemd; het DAN-deel wordt de kop van de regel genoemd. Het gegeven, waarvan in de kop van de regel sprake is, wordt een afgeleid gegeven genoemd, d.i. door de regel afgeleid vanuit de gegevens, waarvan reeds bekend is, dat de waarheidswaarde TRUE is. De verzameling regels wordt de *regelbank* genoemd. De database en de regelbank samen wordt een *kennisbank* genoemd. Hiermee wordt aangesloten bij veel literatuur, waar een kennisbank wordt gedefinieerd als "feiten en

regels" (zie bijvoorbeeld (Waterman, 1986)). De begrippen "gegeven" en "feit" worden in de rest van dit boek als synoniemen beschouwd.

Dat deel van het programma, dat daadwerkelijk de afleiding tot stand brengt wordt het *redeneermechanisme* genoemd.

Het is geen beperking om alleen regels te beschouwen als wijze van kennisrepresentatie. Andere vormen van kennisrepresentatie zoals semantische netwerken en frames, kunnen worden vertaald in een database, een regelbank en een redeneermechanisme.

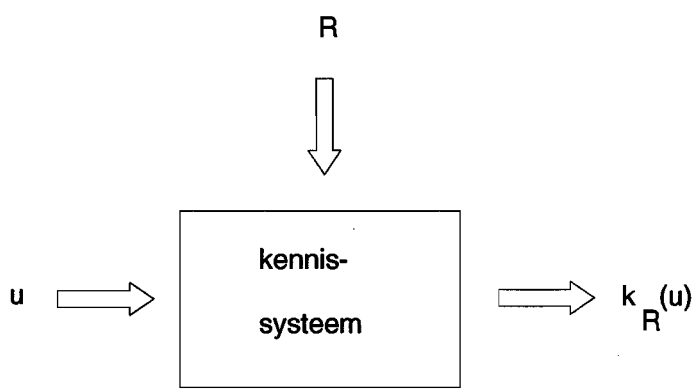
In feite "herkent" een kennissysteem vier verschillende, disjuncte verzamelingen van gegevens:

1. Een verzameling gegevens die aanwezig zijn in een database.
2. Een verzameling gegevens die met behulp van de regels kunnen worden afgeleid vanuit de database en niet al voor het afleidingsproces in de database aanwezig waren.
3. Een verzameling gegevens waarbij ieder gegeven het tegengestelde is van een gegeven uit de verzamelingen bij 1. of 2.
4. Alle overige gegevens.

De vereniging van de verzamelingen onder 1. en 2. wordt de *deductieve afsluiting* van de database onder de regelbank genoemd. Er wordt aangenomen dat alle gegevens, die element zijn van de deductieve afsluiting een waarheidswaarde TRUE hebben. In feite betekent deze aanname dat op ieder moment de database een correcte afspiegeling is van de werkelijkheid, die door die database wordt gemodelleerd. Als deze aanname om wat voor reden dan ook niet correct blijkt te zijn in een specifieke situatie, ontbreekt de basis voor het systeem om correcte redeneringen te maken. Het systeem "weet" dan immers niet wat wel en wat niet als correct mag worden verondersteld.

De verzameling onder 3. wordt de verzameling *verboden uitspraken* genoemd. Als gevolg van de aanname, dat alle gegevens in de deductieve afsluiting een waarheidswaarde TRUE hebben, hebben alle gegevens uit de verzameling verboden uitspraken een waarheidswaarde FALSE. De verzameling onder 4. tenslotte wordt de verzameling *ontoegankelijke uitspraken* genoemd. Het is niet mogelijk om de waarheidswaarde van gegevens uit deze laatste verzameling te bepalen met alleen de database en de regelbank.

De deductieve afsluiting van een database onder een regelbank bevat dezelfde gegevens als *potentieel* aanwezig zijn in de oorspronkelijke database en de regelbank. In feite kan de deductieve afsluiting de *betekenis* van de kennisbank worden genoemd. Er kan worden bewezen, dat bij iedere database precies één deductieve afsluiting hoort, gegeven een regelbank (zie stelling A.1 in bijlage A). Het verband tussen een database en zijn deductieve afsluiting onder een gegeven regelbank kan daarom worden beschreven met behulp van een functie. Deze functie wordt de *kennisfunctie* genoemd. In feite heeft de kennisfunctie slechts een theoretische waarde. In de praktijk is het vanwege de grootte van de database en de regelbank, vaak ondoenlijk om van tevoren voor iedere database het beeld onder de kennisfunctie (de deductieve afsluiting) aan te geven. Met behulp van een kennisfunctie is het mogelijk het kennissysteem als een black box te beschrijven. Zonder te weten HOE een kennissysteem de deductieve afsluiting berekent kan door deze functie worden aangegeven WAT een kennissysteem behoort te berekenen. Schematisch kan dit worden aangegeven zoals in figuur 2.1. Hierin is u de database, R de regelbank en $k_R(u)$ de deductieve afsluiting van u onder R (k is de kennisfunctie).



Figuur 2.1 Black-box voorstelling van een kennissysteem

De trigger voor een kennissysteem om (een deel van) de deductieve afsluiting te gaan berekenen wordt gegeven door een *query*. (De term "query" in deze context is afkomstig uit (Lloyd, 1987) en is synoniem aan "vraag" uit de voorlopige definitie, die aan het begin van deze paragraaf werd gepresenteerd.) Een query bestaat uit een verzameling uitspraken. Een uitspraak in een query wordt een *hypothese* genoemd. Hierbij kunnen twee klassen van queries worden onderscheiden. De ene klasse bevat die queries waarin geen variabelen

voorkomen. De andere klasse bevat die queries waarin een variabele of meer variabelen voorkomen. De queries uit de eerstgenoemde klasse worden *grondqueries* genoemd. Het systeem moet bij een grondquery de waarheidswaarde van deze verzameling bepalen. Hierbij wordt de query opgevat als een conjunctie van de hypothesen die erin voorkomen. Dit wil zeggen dat de query een waarheidswaarde TRUE heeft dan en slechts dan als alle hypothesen in de query een waarheidswaarde TRUE hebben.

Bij het beantwoorden van een grondquery gaat het systeem voor iedere hypothese na in welke van de eerder genoemde verzamelingen van gegevens de hypothese voorkomt. Als een hypothese als element voorkomt in de deductieve afsluiting van de database onder de regelbank, dan heeft het de waarheidswaarde TRUE. Als een hypothese als gegeven voorkomt in de verzameling verboden uitspraken, dan heeft het de waarheidswaarde FALSE. In dit geval heeft ook de gehele query een waarheidswaarde FALSE. Als een hypothese als gegeven voorkomt in de verzameling ontoegankelijke uitspraken dan heeft de hypothese de waarde UNKNOWN. Dit type kennissystemen zal *strikt redenerend* worden genoemd. Omdat dit laatste geval in de praktijk vaak onbevredigend is, wordt aan het kennissysteem vaak een extra redeneerregel toegevoegd waarmee het mogelijk is om in zo'n geval toch een waarheidswaarde TRUE of FALSE te verkrijgen. Zo'n redeneerregel wordt een *externe regel* genoemd. Een kennissysteem dat gebruik maakt van een externe regel zal *uitgebreid redenerend* worden genoemd. Een voorbeeld van een externe regel is de Closed World Assumption (CWA). Deze regel geeft aan alle gegevens uit de verzameling ontoegankelijke informatie de waarheidswaarde FALSE. Andere voorbeelden van externe regels zijn de Herbrand-regel en de Negation as Finite Failure regel (Lloyd, 1987). Door het systeem wordt de waarheidswaarde van de query als antwoord aan de gebruiker gegeven.

Bij het beantwoorden van een niet-grondquery wordt door het systeem gezocht naar alle substituties voor de variabelen, die ervoor zorgen, dat de query een grondquery wordt met een waarheidswaarde TRUE. Dit betekent dat na zo'n substitutie alle in de query voorkomende hypothesen als gegeven moeten voorkomen in de deductieve afsluiting van de database onder de regelbank. Deze verzameling substituties kan leeg zijn. De door het systeem gevonden verzameling substituties wordt als antwoord aan de gebruiker gegeven.

Voorbeeld 2.4.1 geeft een illustratie van de theorie.

VOORBEELD 2.4.1

Stel gegeven de database u en de regelbank R met de volgende inhoud:

u = { vader(Bernhard,Beatrix), vader(Claus,Johan-Friso), ¬moeder(Juliana,Jaime), vrouw(Beatrix) }

R = { vader(x,y)∧vrouw(y) → dochter(y,x) }

Dan geldt:

De deductieve afsluiting is u ∪ { dochter(Beatrix,Bernhard) }

De verzameling verboden uitspraken is:

{ ¬vader(Bernhard,Beatrix), ¬vader(Claus,Johan-Friso), moeder(Juliana,Jaime), ¬vrouw(Beatrix), ¬dochter(Beatrix,Bernhard) }

Beschouw de volgende queries.

a. { dochter(Beatrix,Bernhard) }. Omdat de uitspraak dochter(Beatrix,Bernhard) element is van de deductieve afsluiting van u onder R is het antwoord op deze query TRUE.

b. { moeder(Juliana,Jaime) }. Omdat de uitspraak moeder(Juliana,Jaime) element is van de verzameling verboden uitspraken is het antwoord op deze query FALSE.

c. { vader(X,Beatrix) } (X is een variabele). Omdat vader(Bernhard,Beatrix) de enige uitspraak in de deductieve afsluiting van u onder R is van een analoge vorm als de hypothese, is het antwoord op de query {{X\Bernhard}}. Onder "analoge vorm" moet hier worden verstaan: een gegeven "vader(..)" met als tweede argument "Beatrix". Het antwoord moet worden geïnterpreteerd als: substitueer voor X de waarde "Bernhard" om een gegeven uit de deductieve afsluiting van u onder R te krijgen.

d. { vader(X,Y) }. Omdat vader(Bernhard,Beatrix) en vader(Claus,Johan-Friso) de enige elementen zijn in de deductieve afsluiting van u onder R met een analoge vorm als de hypothese, is het antwoord op de query: {{X\Bernhard,Y\Beatrix},{X\Claus,Y\Johan-Friso}}

e. { vader(Bernhard,Irene) }. Omdat deze hypothese noch als gegeven voorkomt in de deductieve afsluiting van u onder R, noch in de verzameling verboden uitspraken, is het een element van de verzameling ontoegankelijke uitspraken. Het antwoord op deze query is daarom UNKNOWN. Zou door het systeem de CWA gebruikt worden, dan wordt het antwoord door het toepassen van deze externe regel FALSE.

□

Bij het beantwoorden van een grondquery wordt door het systeem een redenering opgebouwd met als uiteindelijke doel van alle hypothesen uit de query de waarheidswaarde te bepalen.

Het systeem berekent daartoe stap voor stap de deductieve afsluiting van de database onder de gegeven regelbank. Zodra daarbij van een hypothese de waarheidswaarde TRUE is aangetoond, doordat het systeem voldoende gegevens van de deductieve afsluiting heeft berekend, wordt de hypothese verwijderd uit de query. Het is ook mogelijk, dat het systeem voor het beantwoorden van een hypothese andere hypothesen aan de query toevoegt. Deze vorm van redeneren wordt wel "backward chaining" of "achterwaarts redeneren" genoemd. Het antwoord van de oorspronkelijke hypothese is dan af te leiden uit de antwoorden op de toegevoegde hypothesen.

Voor het beantwoorden van een niet-grondquery is het systeem in theorie "verplicht" de deductieve afsluiting van de database onder de regelbank te berekenen. Het moet dan immers als antwoord alle substituties geven, die de dan ontstane grondquey een waarheidswaarde TRUE geeft. Anders geformuleerd: het moet dan die substituties geven, die van de hypothesen gegevens uit de deductieve afsluiting van de database onder de regelbank maken. Er zijn echter in veel gevallen mogelijkheden om slechts een deel van de deductieve afsluiting te berekenen. In dit kader zal hier niet verder op worden ingegaan.

Een redenering van het kennissysteem is opgebouwd uit redeneerstappen. Bij een redeneerstap kan de database worden uitgebreid met gegevens, die worden afgeleid met behulp van een regel uit de regelbank. Ook kan de query worden gewijzigd door het verwijderen van beantwoorde hypothesen of het toevoegen van nog te beantwoorden hypothesen. Het effect van iedere redeneerstap kan daardoor worden beschreven met behulp van een *redeneertoestand.* Een redeneertoestand is een paar, bestaande uit de database, die als gevolg van een redeneerstap is ontstaan en de verzameling hypothesen, waarvan de waarheidswaarde na de redeneerstap nog moet worden aangetoond. De totale redenering kan worden beschreven met een rij van redeneertoestanden. Zo'n rij wordt een *redeneerketen* genoemd. Het eerste element van een redeneerketen is altijd het paar, dat bestaat uit de "vaste" database en de query, die aan het systeem is gesteld.

Het weergeven van (een deel van) een redeneerketen kan een implementatie zijn voor de HOW- en WHY-uitlegfunctie. De redeneerketen, die aan het einde van een redenering is ontstaan, beschrijft, op welke wijze het systeem tot het uiteindelijk gepresenteerde antwoord is gekomen. Dit geeft antwoord op de HOW-vraag. De laatste schakel van een redeneerketen, zoals die tijdens een redeneerproces aanwezig is, geeft weer, welke volgende redeneerstap door het systeem wordt "overwogen". Dit geeft het antwoord op de WHY-vraag.

Er zijn diverse redeneerstappen mogelijk. Voor ieder type redeneerstap kunnen de voorwaarden (bijvoorbeeld in de vorm van pre- en postcondities) worden vastgelegd. De

regels, die beschrijven, welke redeneerstappen mogelijk zijn voor een specifiek kennissysteem worden *redeneerregels* genoemd. Logisch correcte redeneerstappen zijn met behulp van vier redeneerregels te beschrijven:

1. De database wordt uitgebreid met een gegeven, dat uit de oorspronkelijke database kan worden afgeleid door middel van een regel uit de regelbank.
2. Een bewezen hypothese wordt verwijderd uit de query.
3. Een bewezen hypothese wordt verwijderd uit de query en als gegeven toegevoegd aan de database.
4. De query wordt uitgebreid met een verzameling hypothesen. Het antwoord op de nieuw toegevoegde hypothesen geeft tevens het antwoord op een hypothese, die reeds in de query aanwezig was.

De redeneerstap, waarbij de database wordt uitgebreid met gegevens uit de verzameling ontoegankelijke uitspraken via een externe regel, wordt beschreven door de volgende redeneerregel:

5. Een hypothese, die een waarheidswaarde TRUE heeft na het toepassen van een externe regel, wordt verwijderd uit de query en als gegeven toegevoegd aan de database.

Voor het kunnen uitvoeren van een redeneerstap heeft een kennissysteem de volgende functies nodig:

a. Een *hypothese-selectiefunctie*. Hiermee wordt uit de query de eerstvolgende hypothese gekozen die door het systeem beantwoord gaat worden.
b. Een *regel-selectiefunctie*. Hiermee wordt een regel uit de regelbank gekozen, waarmee een redeneerstap kan worden gemaakt die nodig is om de gekozen hypothese te beantwoorden.

Redeneerstrategieën, die een expert op een zeker domein ontwikkeld heeft, om zo snel mogelijk tot een optimale oplossing te komen, kunnen vaak vertaald worden in specifieke hypothese- en regel-selectiefuncties. Voorbeeld 2.4.2 verduidelijkt dit.

VOORBEELD 2.4.2
Stel dat de regelbank met regels over familierelaties de volgende twee regels bevat:

a. zoon(X,Y) → kind(X,Y)
b. dochter(X,Y) → kind(X,Y)

(Ofwel: X is een kind van Y als X de zoon van Y is (regel a.) of als X de dochter van Y is (regel b.)).

De redeneerstrategie die stelt, dat bij vragen over kinderen in de Koninklijke Familie beter eerst naar zonen dan naar dochters kan worden gezocht, kan met behulp van de volgende regel-selectiefunctie worden weergegeven:

Als de hypothese van de vorm kind(,,) is
Dan probeer eerst regel a. en vervolgens regel b.

□

De stappen die voor het maken van een redenering moeten worden gezet, staan geprogrammeerd in een *redeneerprocedure* of inferentieprocedure. Deze procedure maakt gebruik van de in het systeem gedefinieerde hypothese- en regel-selectiefuncties en (eventueel) een externe regel die nodig is voor het uitgebreid redeneren. De redeneerprocedure vormt het hart van het redeneermechanisme. Bekende redeneerstrategieën zoals voorwaarts en achterwaarts redeneren (forward resp. backward chaining) kunnen op de volgende wijze worden "vertaald" in een deel van een redeneerprocedure:

BEGIN
{Backward chaining}
ALS een hypothese ter beantwoording is gekozen
DAN zoek die regels uit de regelbank, waarbij de hypothese in de kop van de regel voorkomt

SELECTEER een van de gevonden regels volgens de regel-selectiefunctie.
EINDE

BEGIN
{Forward chaining}
ALS een hypothese ter beantwoording is gekozen
DAN zoek die regels uit de regelbank, waarbij alle uitspraken in de staart van de regel als gegeven voorkomen in de database

SELECTEER een van de gevonden regels volgens de regel-selectiefunctie.
EINDE

Er kan worden bewezen (stelling A.7), dat het antwoord dat door het systeem wordt gegeven, onafhankelijk is van de redeneerprocedure. De redeneerprocedure kan echter wel van invloed zijn op de snelheid, waarmee een antwoord wordt verkregen. Deze snelheid wordt mede bepaald door de inhoud van de hypothese- en regel-selectiefunctie. De eerder gegeven regel-selectiefunctie uit voorbeeld 2.4.2 is hier een voorbeeld van.

Samengevat kan een kennissysteem worden gekarakteriseerd door het *expliciet* aanwezig zijn van de volgende componenten:

1. Een database-deel
   - Een definitie van de verzameling toegelaten databases
   - Een toegelaten database
   - Een controle mechanisme
2. Een regelbank
3. Een redeneerdeel
   - Een externe regel
   - Een verzameling redeneerregels
   - Een hypothese-selectiefunctie
   - Een regel-selectiefunctie
   - Een redeneerprocedure

Expliciet aanwezig moet hier worden opgevat als: afzonderlijk benaderbaar. Het moet dus bijvoorbeeld mogelijk zijn de regelbank apart te benaderen en te bewerken, zonder tijdens het laatste rekening hoeven te houden met gevolgen voor de andere componenten. Indien de regelbank ingebed ligt in een (imperatieve) procedure met meer taken (bijvoorbeeld regel-selectie), dan is duidelijk geen sprake van afzonderlijke benadering. Een systeem, waarbij dit laatste het geval is, is dan ook geen kennissysteem volgens de hier gegeven definitie.

In de rest van dit boek zal het hier gegeven model van een kennissysteem het XYFAR-model worden genoemd. Dit is een acroniem voor KennisSYsteem Formeel ARchitectuur-model (waarbij de eerste twee letters KS zijn vervangen door de letter X).

2.5 OPMERKINGEN BIJ HET XYFAR-MODEL

In paragraaf 2.4.1 werden enkele beperkingen geformuleerd voor het uiteindelijk geformuleerde XYFAR-model. Een van die beperkingen was, dat in het model niet werd uitgegaan van onzekere kennis. In het model kunnen echter onzekerheidsberekeningen, die tijdens een redenering worden gedaan, ingepast worden, zonder dat daardoor het model moet worden aangepast. Een feit, dat (op een schaal van 0 tot 1) een onzekerheidswaarde 0 heeft, krijgt als waarheidswaarde FALSE toegemeten. Alle andere feiten krijgen waarschijnlijkheidswaarde TRUE. Natuurlijk wordt, parallel aan de berekeningen binnen het kennissysteem, de onzekerheidswaarde van ieder feit wel bijgehouden. Het rekening houden met een onzekerheidswaarde kan gevolgen hebben voor de keuze van een hypothese of de keuze van een regel. Bij de hypothese- en regelselectiefunctie kan daar rekening mee worden gehouden. Een (informeel gesteld) voorbeeld van een regel-selectiefunctie zou kunnen zijn: "Kies die regel, waarbij een feit geconcludeerd wordt met de hoogste onzekerheidswaarde". Het presenteren van de uitkomst met daaraan toegevoegd een onzekerheidswaarde is in dit model het presenteren van het (uitgebreide) antwoord (waarbij het antwoord geen waarschijnlijkheid krijgt toegemeten) en de bijbehorende waarschijnlijkheidswaarde (wat buiten het gepresenteerde model valt).

In het model en zijn werking wordt globaal het volgende beeld geschetst van een kennissysteem. Als een kennissysteem een vraag (query) krijgt, probeert het een antwoord te vinden door de inhoud van de database te inspecteren en (eventueel) de deductieve afsluiting te berekenen. Als die berekening is gebeurd kan een antwoord worden gegenereerd. Echt "intelligent" is dit gedrag niet, getuige het volgende voorbeeld.

VOORBEELD 2.5.1
Stel gegeven een DB-toestand
u = {getrouwd(Jorge,Christina), getrouwd(Pieter,Margriet)}
en de constraint c, geformuleerd door:
$\forall$x,y,z: getrouwd(x,z) $\wedge$ getrouwd(y,z) $\rightarrow$ x=y (ieder persoon is met ten hoogste een ander persoon getrouwd). Stel verder gegeven een willekeurige regelbank R. Als aan deze kennisbank de query { getrouwd(Pieter,Christina) } wordt aangeboden, zal het systeem

(doordat de hypothese niet in "u" voorkomt) $k_R(u)$ berekenen en vervolgens met een externe regel de waarheidswaarde bepalen. Als het systeem echter in staat zou zijn de constraint bij zijn redenering te betrekken, dan zou het onmiddellijk (uit de DB-toestand) kunnen concluderen, dat het antwoord FALSE moet zijn.
□

Zulk intelligent gedrag treedt niet op, doordat kennissystemen volgens het XYFAR-model uitgaan van de feiten en niet van de struktuur van de regels en constraints. Zou in een redeneerprocedure allereerst naar constraints gekeken worden, dan zou een redenering als volgt kunnen verlopen (met voorbeeld 2.5.1 als uitgangspunt):

- getrouwd(Pieter,Christina) is geen element van u.
- getrouwd(Jorge,Christina) is element van u (de waarde van het tweede argument komt overeen met de waarde van het tweede argument uit de query).
- doe de substitutie { x\Jorge,y\Pieter,z\Christina} bij de formulering van c. Er onstaat dan een onware uitspraak. Conclusie: antwoord is FALSE.

Door het toestaan van dit soort redeneringen moet in het model de definitie van (uitgebreid) antwoord worden aangepast. Dit wordt in het model verder niet meegenomen.

Uit de presentatie van het XYFAR-model zou kunnen worden geconcludeerd dat het model alleen uitgaat van stand-alone kennissystemen. Die conclusie is echter onjuist. Een kennissysteem kan ook deel uitmaken van een groter informatiesysteem. Hierbij is de databasecomponent uit het XYFAR-model te beschouwen als een externe view van de database die door het totale informatiesysteem wordt gebruikt. Een andere mogelijkheid voor *integratie* van een kennissysteem in een groter informatiesysteem ontstaat, indien niet alleen de database "geshared" wordt door meerdere (deel)systemen maar ook de regelbank door meerdere applicaties tegelijk wordt gebruikt. Iedere applicatie die gebruik maakt van de regelbank, doet dit via een externe view op die regelbank. Hierbij is het begrip "externe view op een regelbank" analoog aan het begrip "externe view op een database" vanuit de 3-schema-architectuur voor databases. Door de externe view op de regelbank te beschouwen als "de" regelbank voor een applicatie, kan het XYFAR-model ook deze situatie beschrijven. In hoofdstuk 6 wordt nader op deze situaties ingegaan.

2.6 VOORBEELD: MYCIN

Om het hiervoor geschetste model te verduidelijken, zal een voorbeeld worden gegeven van een invulling van de onderscheiden componenten, zoals die in een bestaand (kennis)systeem zijn te vinden. Het voorbeeld betreft het "klassieke" kennissysteem MYCIN, waarvan de beschrijving werd gehaald uit (Buchanan en Shortliffe, 1984). In de beschrijving zal meerdere malen naar dit boek worden verwezen door het aangeven van paginanummers op deze wijze: (B&S, p. xx).

De beschrijving van MYCIN moet de lezer tevens voldoende overtuigen van de "rijkheid" van het XYFAR-model. Hiermee wordt bedoeld, dat de redeneeraspecten van MYCIN kunnen worden beschreven met behulp van de terminologie uit het XYFAR-model.

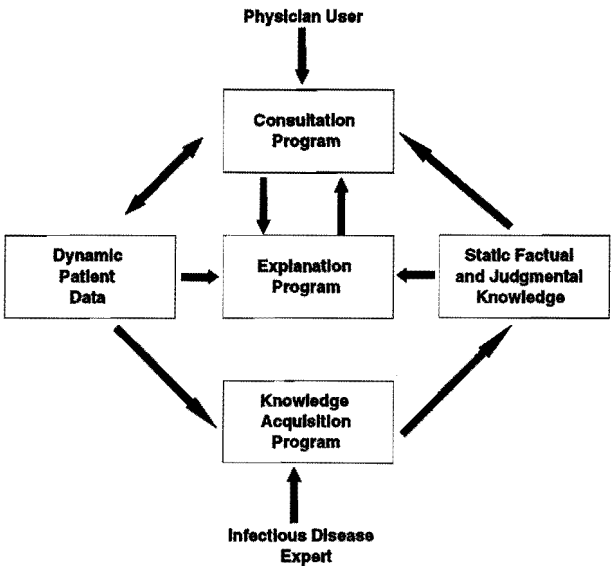
MYCIN is een kennissysteem, dat in de 70-er jaren werd ontwikkeld. Het domein van kennis is dat van diagnose van infectieziekten en het vervolgens aangeven van een therapie voor de genezing van zo'n ziekte.

De architectuur van MYCIN wordt getoond in figuur 2.2 (naar (B&S, p. 68)). De pijlen in deze figuur geven een informatiestroom aan tussen componenten van het systeem. (In deze figuur is een informatiestroom toegevoegd vanuit het Explanation Program naar het Consultation Program. Zonder deze pijl heeft het Explanation Program geen nut. Vermoedelijk is dit een tekenfout in het boek van Buchanan en Shortliffe.)

De kern van dit kennissysteem, zoals beschreven in het XYFAR-model, bestaat hier uit:

1. Dynamic Patient Data.
2. Static Factual and Judgmental Knowledge.
3. Consultation Program. Deze bevat onder meer de Rule Interpreter.

De in het model onderscheiden componenten zijn echter niet één op één af te beelden op de hier onderscheiden systeemcomponenten. Dit zal verderop blijken. Grofweg kan de volgende afbeelding worden gemaakt:



Figuur 2.2 Architectuur van MYCIN

Database → Dynamic Patient Data en Static Factual Knowledge
Regelbank → Judgmental Knowledge
Hypothese-selectiefunctie → Judgmental knowledge en Consultation Program
Regel-selectiefunctie → Judgmental knowledge en Consultation Program
Redeneerregels → Consultation Program (meer specifiek de Rule Interpreter)
Externe regel → Judgmental Knowledge
Redeneerprocedure → Consultation Program

Zoals uit deze globale opsomming al blijkt, zijn de componenten uit het XYFAR-model niet expliciet terug te vinden in de architectuur van MYCIN. Het is echter wel mogelijk alle redeneer-functionaliteiten van MYCIN te beschrijven in termen van het XYFAR-model.

2.6.1 MYCIN EN HET XYFAR-MODEL

Hierna volgt een beschrijving van in welke vorm de componenten uit het XYFAR-model zijn terug te vinden in de architectuur van MYCIN.

DATABASE

Het systeem gebruikt tripels <object,attribuut,waarde> voor de representatie van de data. Data over ziektebeelden en therapieën behoren tot de "vaste" database. Data over een specifieke patiënt worden opgevraagd tijdens consultatie van het systeem. Deze laatste actie is in het XYFAR-model niet gekenmerkt als een soort externe regel, maar als een aanvulling van de database. De tijdens consultatie opgevraagde data is immers wel gedefinieerd in een DB-universum. Een latere versie van MYCIN had overigens ook de mogelijkheid patiëntendata te bewaren voor toekomstige consultaties.

REGELBANK

Binnen MYCIN is een ordening in de regelbank aangebracht. Tevens bevat de regelbank diverse typen regels, ieder met een eigen doel. De voornaamste afwijking binnen MYCIN ten opzichte van het model is de aanwezigheid van zgn. Self-referencing rules. Dit zijn regels voor:

a. Default reasoning. Een voorbeeld van zo'n regel is:

   ALS de waarde van X niet kan worden afgeleid (UNKNOWN)
   DAN is de waarde van X gelijk aan z.

   Dit type regel is een "gewone" regel uit een regelbank volgens het XYFAR-model. Het kan echter een regel-selectiefunctie vereisen, die deze regel als laatste probeert.

b. Screening-taak: niet vragen naar gegevens, tenzij er al aanleiding was om de hypothese te overwegen.

   Bijvoorbeeld:

   De regel A $\wedge$ B $\wedge$ C -> A (*) heeft de betekenis: "als A met (voldoende) zekerheid x mag worden geconcludeerd en B en C mogen ook met voldoende zekerheid worden geconcludeerd, dan mag A met zekerheid y worden geconcludeerd, waarbij y > x". Dit is eveneens een voorbeeld van een regel-selectiefunctie. Hierdoor wordt vermeden, dat naar B en C wordt gevraagd, terwijl dit achteraf niet nodig is. Dit is handig als vragen naar B en C betekent, dat er veel werk wordt gedaan, dat je zoveel mogelijk wil vermijden. Omdat het systeem een backward-chaining strategie gebruikt wordt de regel in het voorbeeld pas in beschouwing genomen, als er al voldoende zekerheid is

over A. Door regel (*) te vervangen door de regel B $\wedge$ C -> A en een regel-selectiefunctie toe te voegen, die de laatstgenoemde regel niet als eerste gaat proberen, wordt hetzelfde effect bereikt. (Een soortgelijke opmerking is te vinden op (B&S, p. 394)).

EXTERNE REGEL

Hoewel niet expliciet aangegeven, wordt in de geraadpleegde literatuur melding gemaakt van een "gesloten wereld" : "... the MYCIN world can be considered to be closed,..." (B&S, p. 469). Dit suggereert dat het systeem bij redeneringen daadwerkelijk gebruik maakt van de CWA als externe regel.

REGEL-SELECTIEFUNCTIE

Het systeem maakt impliciet gebruik van de regel-selectiefunctie:

Selecteer de regels in volgorde van voorkomen in de regelbank

Het systeem kent echter diverse andere regel-selectiefuncties. De volgende vier worden expliciet genoemd (B&S, p. 523)

Metarule001

IF the culture was not obtained from a sterile source
AND there are rules which mention in their premise a previous organism which may be the same as the current organism
THEN it is definite (1.0) that each of them is not going to be useful

Metarule002

IF the infection is a pelvic-abscess
AND there are rules which mention in their premise enterobacteriaceae
AND there are rules which mention in their premise gram-positive rods
THEN there is suggestive evidence (.4) that the former should be done before the latter

Metarule003

IF there are rules which do not mention the current goal in their premise
AND there are rules which mention the current goal in their premise
THEN it is definite that the former sould be done before the latter

Metarule004

IF there are rules which are relevant to positive cultures
AND there are rules which are relevant to negative cultures
THEN it is definite that the former should be done before the latter

Merk op, dat metarule003 domein-onafhankelijk is, terwijl de overige metarules domeinafhankelijk zijn.

Tevens kent MYCIN een zgn. *preview-mechanisme* (B&S, p. 493). Dit houdt in, dat MYCIN, voordat een regel gebruikt gaat worden eerst alle premissen van de regel bekijkt (zonder een redenering te starten) om te zien, of de waarheidswaarde van een van de premissen FALSE is. Als dat het geval is, dan wordt de regel niet in beschouwing genomen.

HYPOTHESE-SELECTIEFUNCTIE

In het systeem wordt geen expliciet gebruik gemaakt van een hypothese-selectiefunctie. Wel suggereert de volgende passage op (B&S, p. 395) een hypothese-selectiefunctie, die gebruik maakt van de tekstuele volgorde:

> "Another means of influencing the control is to order the clauses in premises of rules. (...) Since MYCIN evaluates the premise clauses from first to last, in order, putting more general, context-setting clauses at the beginning of the premise assures that the more specific clauses will not be asked about, or even considered, unless the context is appropriate. Using the order of premise clauses for this kind of screening permits the system builder to use early clauses to ensure that some parameters are traced first."

INFERENTIEPROCEDURE

De basisstrategie, die door MYCIN wordt gebruikt is de backward-chaining strategie. De eerder vermelde strategieregels uit de regelbank (self-referencing rules, metarules) kunnen echter gebruikt worden om het redeneerproces efficiënt te laten verlopen en in een vroeg stadium van de redenering doodlopende wegen te onderkennen. Andere controle-structuren, die expliciet worden gebruikt door de inferentieprocedure zijn:

- Een *goalrule* (ook wel rule092 genoemd). Deze regel geeft de taak, die door het systeem moet worden opgelost op het hoogste niveau van abstractie weer. De regel luidt (B&S, p. 554):

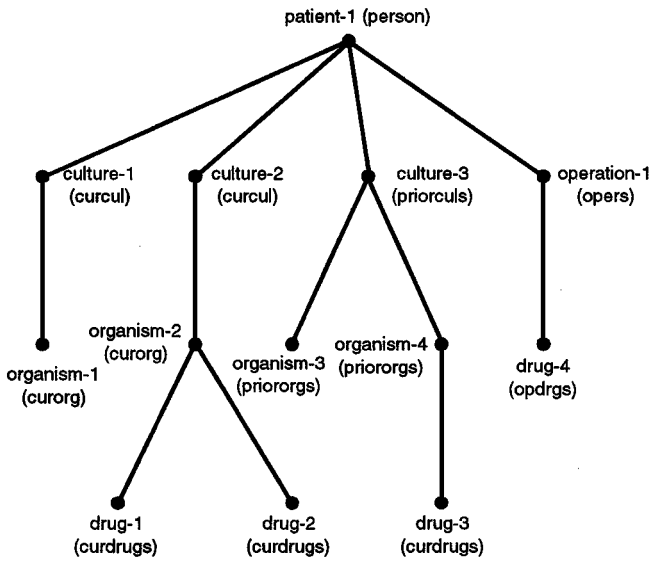
  IF Gather information about cultures taken from the patient and therapy he is receveing
  Determine if the organisms growing on cultures require therapy
  Consider circumstantial evidence for additional organisms that therapy should cover
  THEN Determine the best therapy recommendation

  Het redeneerproces wordt gestart door het aanroepen van de goalrule.
- Een *contextboom* (B&S, pp. 494 ff). De contextboom bevat kennis over de relaties tussen de objecten, die in de regels worden genoemd. Een voorbeeld van een contextboom uit MYCIN is te vinden in figuur 2.3 (B&S, p. 84). In deze figuur staat de contextboom voor een patiënt met 2 recente positieve bacteriekweken (resp. CULTURE-1 en CULTURE-2) en 1 oudere positieve kweek (CULTURE-3). Tevens heeft de patiënt recent een operatie van belang ondergaan (OPERATION-1). Iedere knoop in de boom is een instantiatie van een context. De naam van de betreffende context staat tussen haakjes bij de knoop. Voor iedere regel uit de regelbank staat aangegeven, in welke contexten ze gebruikt kunnen worden. In feite legt de contextboom zo een structuur over de regelbank.

Tenslotte moet nog worden opgemerkt, dat MYCIN gebruik maakt van onzekerheidsfactoren bij het redeneren. Dit heeft onder meer de volgende gevolgen:

- Het systeem zal *alle* conclusies presenteren, die met een onzekerheid groter dan 0.20 kunnen optreden. Dit geeft geen afwijking van het beeld, dat in het XYFAR-model is geschetst. Gegevens, die afgeleid worden, zijn element van de deductieve afsluiting van de database onder de gegeven regelbank. Het maakt daarbij niet uit, of een

  attribuut van dat gegeven een zekerheidsfactor is.
- Voordat geredeneerd gaat worden, wordt door het systeem eerst gezocht naar een regel of een keten van regels, die met zekerheid 1.0 leidt tot de gewenste conclusie. Als er zo'n keten bestaat (in MYCIN een *unity path* genoemd), dan wordt deze als eerste geprobeerd. Slagen van zo'n redenering betekent namelijk dat alternatieve redeneerpaden niet meer hoeven te worden nagegaan. Dit unity path algoritme kan worden beschouwd als een regel-selectiefunctie.



Figuur 2.3 Voorbeeld van een contextboom

- MYCIN staat regels toe van de vorm "conditie $\rightarrow A_1\ (c_1) \vee A_2\ (c_2) \vee$...". In woorden: als "conditie" waar is, mag A_1 met zekerheid c_1 worden geconcludeerd of A_2 met zekerheid c_2 of.... Dit type regels valt echter buiten het beschouwingsgebied van het XYFAR-model. Daarin werden geen regels toegestaan met onvolledige informatie in de kop.

2.6.2 EVALUATIE

Zoals in paragraaf 2.6.1 werd beschreven kunnen alle redeneeraspecten van MYCIN, die niet te maken hebben met het beschouwen van onvolledige informatie, beschreven worden in de terminologie van het XYFAR-model. Dit maakt het aannemelijk, dat het model voldoende "rijk" is. Strikt genomen is MYCIN volgens het XYFAR-model echter geen kennissysteem. De volgende redenen zijn daarvoor aan te voeren:

- Door expliciet de componenten in het XYFAR-model te onderscheiden worden strategieën expliciet gemaakt. Binnen MYCIN gebeurt dit echter niet overal. Vooral de hypothese-selectiefunctie is impliciet gelaten. Dit gaat ten koste van de

duidelijkheid van de regelbank, omdat bij de formulering van de regels de bouwer bekend moet zijn met de wijze, waarop MYCIN de verzameling hypothesen evalueert.

- De regelbank bevat een mengeling van regels en redeneerstrategieën. De genoemde goal-rule is daar een voorbeeld van. Nog sterker geldt dit voor de self-referencing regels. Daar wordt binnen één regel redeneerstrategie en afleidingskennis gebruikt. Tevens wordt bij de formulering van de regel impliciet gebruik gemaakt van de (eveneens impliciet gelaten) hypothese-selectiefunctie. Ook dit gaat ten koste van de duidelijkheid van het systeem.
- Hoewel zowel het unity-path algoritme als het preview-mechanisme expliciet genoemde strategieën zijn binnen MYCIN, wordt hierover tevens opgemerkt, dat deze strategieën gecodeerd zijn in de rule-interpreter (B&S, p. 494). Hierdoor zijn ze niet toegankelijk voor de gebruiker of de beheerder van het systeem. Dit wordt als grootste verschil genoemd met de eerder gegeven metarules.

Het kunnen nagaan op welke wijze redeneeraspecten verwerkt zijn in een kennissysteem, cq. welke invulling de onderkende redeneercomponenten hebben, lijkt een goed gebruiksdoel op te leveren voor het XYFAR-model. Hoofdstuk 3 zal hier dieper op ingaan.

3 CONCEPTUEEL SPECIFICEREN VAN KENNISSYSTEMEN

De beschrijving van MYCIN in termen van het XYFAR-model was een eerste aanwijzing, dat het model gebruikt kan worden als een leidraad bij het maken van een conceptuele specificatie van een kennissysteem. Dit onderwerp zal in dit hoofdstuk nader worden uitgewerkt. Allereerst zal in paragraaf 3.1 worden beschreven, wat in dit boek zal worden verstaan onder een conceptuele specificatie van een informatiesysteem in het algemeen en van een kennissysteem in het bijzonder. Vervolgens zullen in paragraaf 3.2 twee standpunten betreffende de inhoud van een conceptuele specificatie van een kennissysteem worden beschreven, zoals die zijn beschreven in twee methoden voor de ontwikkeling van dit type systemen (KADS en DESIRE). Voor deze twee methoden is gekozen, omdat deze in Nederland veelvuldig worden gebruikt bij de ontwikkeling van kennissystemen. Paragraaf 3.3 geeft aan, hoe het XYFAR-model gebruikt kan worden bij het maken van een conceptuele specificatie en welke voor- en nadelen dit heeft ten opzichte van de twee methoden uit paragraaf 3.2.

3.1 ONTWIKKELEN VAN KENNISSYSTEMEN

Bij het ontwikkelen van een informatiesysteem worden diverse fasen doorlopen. Iedere fase levert een aantal specifieke produktdocumenten op, waarbij bij iedere fase voortgebouwd wordt op de produktdocumenten van voorgaande fasen. In (Aerts et al, 1991) wordt de volgende fasering met bijbehorende produktdocumenten aangehouden:

Fase	Produkt
Informatieplanning	Omgevingsanalyse, informatieplan
Definitiestudie	Eisendocument
Conceptueel ontwerp	Conceptueel model
Implementatie ontwerp	Implementatiemodel, handleidingen, testmodel
Realisatie	Code, handleidingen, gewijzigde produktdocumenten
Invoering	Gewijzigde produktdocumenten
Gebruik en beheer	Gewijzigde produktdocumenten

De term "conceptueel model" is een synoniem voor "conceptuele specificatie". In het vervolg zal deze laatste term worden gebruikt.

Omdat veel wordt voortgebouwd op de conceptuele specificatie is de fase "Conceptueel Ontwerp" een belangrijke fase. De conceptuele specificatie is het eerste volledige systeembeeld qua functionaliteit dat wordt gemaakt. Het bevat een specificatie van het systeem op een zodanige wijze, dat iedere realisatie ervan voldoet aan de functionele en prestatie-eisen, die in de fase Definitiestudie werden geformuleerd. Deze systeemspecificatie is onafhankelijk van de uiteindelijke implementatie opgesteld. De conceptuele specificatie bevat onder meer de volgende twee aspectmodellen:

1. Database-ontwerp. Hierin wordt een beschrijving gegeven van het DB-universum.
2. Functie-ontwerp. Hierin worden de gebruikersfuncties geformuleerd. Een onderdeel van het functie-ontwerp is een dialoogmodel. Hierin wordt de specificatie gegeven van de dialoog, die het systeem voert met de gebruiker.

Deze definitie van een conceptuele specificatie is in lijn met die uit de ANSI/IEEE-standaard (ANSI, 1984). Daar wordt de term "requirements specification" (als synoniem van conceptuele specificatie te beschouwen) als volgt gedefinieerd (ANSI, 1984, p. 30):

> A specification that sets forth the requirements for a system or system component;(..) Typically included are functional requirements, performance requirements, interface requirements, design requirements, and development standards.

Het begrip "specification" wordt als volgt gedefinieerd (ANSI, 1984, p. 33):

> A document that prescribes, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or system component.

Indien de conceptuele specificatie een software-produkt betreft, wordt in de ANSI-standaarden gesproken van een Software Requirements Specification (SRS). In het kader van dit boek zullen de termen conceptuele specificatie, SRS en Requirements Specification als synoniemen worden beschouwd, omdat het onderwerp van dit boek een software-produkt is.

Een conceptuele specificatie heeft onder meer de volgende doelstellingen (ANSI, 1984):

- Het vormen van een basis voor overeenstemming tussen de opdrachtgevers en de leveranciers over de functionaliteit van het software-produkt.
- Het verminderen van de ontwikkelinspanning. Door een conceptuele specificatie nauwkeurig te beschouwen, kunnen tekortkomingen worden opgespoord voordat met de bouw van het systeem is begonnen.

- Het vormen van een basis voor validatie en verificatie van het te ontwikkelen software-produkt.
- Het dienen als een startpunt voor latere aanpassingen aan het produkt.

Een conceptuele specificatie wordt weergegeven met behulp van een specificatietaal. Deze taal bevat taalelementen, waarmee het mogelijk is een specificatie te beschrijven. Als bijvoorbeeld als specificatietaal wordt gekozen voor het Entiteit-Relatie-Diagram, dan zijn de taalelementen de rechthoeken, waarmee entiteittypen kunnen worden weergegeven; de "wybertjes" voor het weergeven van de relatietypen en de lijnstukken voor het weergeven van relaties tussen entiteittypen en/of relatietypen.

Afgeleid van de doelstellingen kunnen eisen worden geformuleerd, waaraan een conceptuele specificatie moet voldoen (ANSI, 1984):

1. De specificatie moet *ondubbelzinnig* zijn. Dit betekent, dat iedere formulering slechts één interpretatie heeft.
2. De specificatie moet *volledig* zijn. Dit betekent, dat alle significante eisen ten aanzien van functionaliteit, performance en externe interfaces aanwezig moeten zijn. Tevens moeten alle systeemreacties voor alle te realiseren invoergegevens gespecificeerd zijn.
3. De specificatie moet *verifieerbaar* zijn. Dit betekent, dat iedere eis zodanig is geformuleerd, dat er een eindig, efficiënt proces bestaat om de realisatie van de eis in het ontwikkelde produkt te kunnen nagaan.
4. De specificatie moet *consistent* zijn. Dit betekent, dat er geen tegenstrijdige aspecten in de specificatie mogen voorkomen.
5. De specificatie moet *onderhoudbaar* zijn. Dit betekent, dat de struktuur en stijl zodanig zijn dat veranderingen aan de specificatie op eenvoudige wijze kunnen worden gemaakt.
6. De specificatie moet *controleerbaar* zijn. Dit betekent, dat de bron van ieder van de eisen duidelijk is.
7. De specificatie moet *bruikbaar* zijn tijdens de fase gebruik en onderhoud.

Afgeleid van de eisen aan een conceptuele specificatie kunnen eisen worden geformuleerd, waaraan een conceptuele specificatietaal moet voldoen:

1. De taal moet voldoende *uitdrukkingskracht* bezitten voor het weergeven van alle aspecten van de specificatie. Dit betekent, dat het met behulp van de ter beschikking staande taalelementen mogelijk moet zijn alle van belang zijnde aspecten te kunnen weergeven, zodat een volledige specificatie kan worden verkregen (eis 2).

2. De taal moet een *eenduidige* specificatie mogelijk maken. Dit betekent, dat ieder element van de specificatie ondubbelzinnig moet kunnen worden geïnterpreteerd. Dit bevordert de consistentie van de specificatie (eis 4).
3. De taal moet mogelijkheden geven tot het *structureren* van de specificatie. Dit betekent, dat het met behulp van de taal mogelijk moet zijn een of andere (bijvoorbeeld hiërarchische) structuur aan te brengen in de specificatie. Zeker bij wat omvangrijkere specificaties geeft dit een goed overzicht over en inzicht in de specificatie. Dit bevordert tevens de onderhoudbaarheid en herbruikbaarheid van een specificatie (eis 5 en eis 7).
4. De taal moet mogelijkheden geven voor het omgaan met *onzekerheid* en *onvolledigheid.* In het bijzonder moet het mogelijk zijn om tijdelijk onvolledige modellen van het beschouwde gebied (Universe of Discourse) te kunnen weergeven. Dit is onontkoombaar tijdens het ontwikkelproces. De taal moet echter wel de mogelijkheden geven onzekerheid en onvolledigheid te kunnen weergeven in de specificatie, omdat zonder deze aspecten een specificatie niet volledig zal zijn (eis 2).
5. De taal moet mogelijkheden geven tot het specificeren van zowel (wiskundig) *formele* als *niet-formele* aspecten (eisen 1, 3 en 7).

Eis 6 (de mogelijkheid de specificatie na te trekken) is meer afhankelijk van de wijze, waarop een specificatietaal wordt gebruikt (de gebruikte stijl van weergeven) en minder van de taal zelf. De stijl van weergeven zal echter ook van invloed zijn op andere eisen (bijvoorbeeld eis 3).

De idee, om een kennissysteem te specificeren onafhankelijk van te gebruiken hard- en software, is onder meer terug te vinden in de methoden KADS en DESIRE. In de volgende paragraaf zal de wijze van specificeren van een kennissysteem binnen die methoden worden besproken. In het bijzonder zal daarbij worden ingegaan op de opvattingen van de beide methoden over wat een conceptuele specificatie van een kennissysteem moet inhouden.

3.2 BESTAANDE METHODEN: KADS EN DESIRE

3.2.1 KADS

KADS is een acroniem voor Knowledge Acquisition and Documentation Structuring. Het is ontwikkeld in het kader van ESPRIT-project P1098. Een uitgebreide beschrijving van KADS is te vinden in (Breuker et al, 1987) en (Breuker en Wielinga, 1991). Op dit moment wordt

er gewerkt aan een verbeterde versie van KADS, wederom in het kader van een Esprit-project (met nummer P5248). Deze uitbreiding staat bekend als KADS-II of CommonKADS.

Een van de uitgangpunten van KADS is pas te starten met de bouw van een kennissysteem als de conceptuele specificatie volledig is. In het bijzonder moet er een model van de in het systeem gebruikte kennis zijn beschreven. In feite is dit een herformulering van het adagium dat bij iedere methode van gestructureerd ontwikkelen van informatiesystemen wordt gehuldigd.

Binnen KADS wordt een *conceptueel model* beschouwd als een specificatie van de kennis, die nodig is om een kennissysteem te bouwen. Deze specificatie wordt gebruikt zowel tijdens het ontwerp als tijdens de bouw van het kennissysteem. Het conceptueel model van een kennissysteem bestaat uit twee deelmodellen: het *model of expertise* en het *model of cooperation.*

In het model of expertise wordt de kennis in het te ontwikkelen kennissysteem beschreven. Dit gebeurt in vier lagen.

1. Domeinlaag. Hierin wordt de statische kennis betreffende het domein beschreven. Deze kennis is taak-neutraal, dwz. dat het niet geformuleerd is in relatie tot enige functie van het systeem. Onder meer wordt in deze laag een beschrijving gegeven van de te onderscheiden objecten in het kennisdomein en hun onderlinge relaties.
2. Inferentielaag. Hierin worden de redeneringen beschreven, die gemaakt kunnen worden met behulp van de kennis uit de domeinlaag. Deze beschrijving wordt gegeven met behulp van twee typen componenten: *kennisbronnen* en *metaklassen.* Een kennisbron beschrijft, welk type redenering kan worden gemaakt op basis van relaties uit de domeinlaag. Een metaklasse fungeert als input of output van een kennisbron en beschrijft de rol van een element uit de domeinlaag in een redenering. Een redenering op dit niveau is te beschouwen als een elementaire bouwsteen voor een redeneerproces. Door de output van een kennisbron als input voor een andere kennisbron te gebruiken ontstaat een netwerk van kennisbronnen, verbonden door metaklassen. Zo'n netwerk wordt een redeneerstructuur genoemd.
3. Taaklaag. Hierin wordt de kennis beschreven over hoe de beschreven redeneringen uit de inferentielaag kunnen worden gebruikt voor het oplossen van een taak. Dit wordt bereikt door een *taakstruktuur* te beschrijven. Een taakstructuur geeft aan, hoe een redeneerstructuur uit de inferentielaag moet worden doorlopen. Twee bekende taakstructuren zijn het forward chaining en het backward chaining principe.

4. Strategielaag. Hierin wordt de kennis beschreven, die het systeem in staat stelt de taaklaag te sturen. Zo kunnen, afhankelijk van de context, taakstructuren uit de taaklaag in verschillende volgorden geprobeerd worden voor het oplossen van een taak. Het bepalen van de volgorde gebeurt dan met behulp van kennis uit de strategielaag.

In KADS-II is het model of expertise gewijzigd (De Hoog, 1992). De strategielaag wordt hier niet meer onderscheiden. In plaats van "kennisbronnen" en "metaklassen" wordt in KADS-II gesproken over "primitieve inferentie acties" en "dynamische kennisrollen". De rol en betekenis van beide concepten zijn echter niet gewijzigd.

Het model of cooperation beschrijft een decompositie van de uit te voeren taak (in de reële wereld) in een aantal primitieve taken. Een primitieve taak wordt daarbij beschouwd als een taak die uit te voeren is door een afzonderlijke "agent". Voorbeelden van agenten zijn het systeem, een gebruiker of een extern systeem. Tevens wordt de samenwerking beschreven die tussen subtaken kan bestaan. Uiteindelijk levert het model of cooperation een beschrijving van de wijze, waarop de gebruiker en het systeem gezamenlijk een doel bereiken, rekening houdend met diverse beperkingen, die kunnen spelen, zoals de taakomgeving, de gebruiker en de state of the art van de beschikbare technologie.

De technieken die binnen KADS worden aangegeven voor de beschrijving van de modellen, zijn niet gebaseerd op een formalisme. Voor de beschrijving van een interpretatiemodel is een eigen tekentechniek ontwikkeld, met vaste symbolen voor kennisbronnen en metaklassen. Tevens wordt gebruik gemaakt van pseudocode voor de beschrijving van de taakstrukturen.

De vier lagen uit het model of expertise kunnen als volgt worden afgebeeld op de componenten uit het XYFAR-model:

Domeinlaag	→	Database-deel, regelbank
Inferentielaag	→	Redeneerregels, selectiefuncties
Taaklaag	→	Redeneerprocedure, selectiefuncties, externe regel
Strategielaag	→	Redeneerprocedure, selectiefuncties, externe regel

3.2.2 DESIRE

De ontwerpmethode DESIRE werd ontwikkeld aan de Vrije Universiteit te Amsterdam. Een doel van DESIRE was het dichten van de kloof die er bestaat tussen de ontwerpresultaten

van een kennisanalyse (zoals die bijvoorbeeld met behulp van de methode KADS worden verkregen) en een daadwerkelijke implementatie van een systeem (Brumsen et al, 1990). Deze kloof heeft tot gevolg dat veel ontwerpkeuzes tijdens de implementatie gemaakt moeten worden. Een voorbeeld hiervan kan de keuze voor de dialoogstruktuur en -vorm betreffen op het moment van de implementatie. Een ander gevolg van de genoemde kloof is de moeilijkheid om een meest passende tool te vinden voor de implementatie van een specifiek systeem. De ontwerpresultaten van een kennisanalyse zijn daartoe niet geschikt.

DESIRE is een acroniem voor DEsign and Specification of Interacting REasoning modules. Uit de naam is af te leiden dat in de visie van de makers van deze methode een kennissysteem bestaat uit een aantal modulen, die een onderling verband hebben. Binnen DESIRE wordt daarom consequent gesproken over *modulaire kennissystemen*. Een module kan daarbij een redeneermodule of een conventionele module zijn (bijvoorbeeld een rekenmodel). Een redeneermodule onderscheidt zich van een conventionele module door zijn redeneervermogen. Iedere redeneermodule bestaat uit:

- een inferentie-mechanisme
- een kennisbank
- een datastruktuur (voor de representatie van de verkregen informatie). Deze datastruktuur wordt ook wel het informatieschema van de module genoemd.

Indien een module geactiveerd wordt zal deze met behulp van zijn inferentiemechanisme en zijn kennisbank proberen de waarheid of onwaarheid van uitspraken af te leiden. Dit wordt de *verfijning* van de informatietoestand genoemd. Het dan verkregen resultaat wordt de deductieve afsluiting van de informatietoestand van de kennisbank genoemd.

Modulen kunnen zich op twee niveaus bevinden: het *objectniveau* en het *systeemniveau* (ook wel metaniveau genoemd). Modulen op het systeemniveau zijn in staat te redeneren over de toestand van modulen op objectniveau. Het activeren van modulen op objectniveau gebeurt dan ook vanuit het systeemniveau.

Welke modulen elkaar wanneer opvolgen wordt op dat niveau vastgelegd in de controlestruktuur. De communicatie tussen modulen gebeurt door middel van *transformaties*. Dit zijn procedures die op basis van de deductieve afsluiting van de informatietoestand van de kennisbank van één module het informatie-schema van een andere module instantieert. Zo'n instantiatie houdt in het aangeven van de waarheidswaarden van de uitspraken in de informatietoestand.

Een specificatie van een kennissysteem volgens DESIRE bevat de volgende onderdelen:

0 de naam van het kennissysteem
1 een of meer modulespecificaties
2 een of meer transformaties
3 controlestruktuur

Ad 1.
Zoals eerder al werd aangegeven worden twee soorten modulen onderscheiden: een redeneermodule en een conventionele module. Een specificatie van een redeneermodule onderscheidt zich daarbij van die van een conventionele module door de aanwezigheid van een kennisbank. Een kennisbank bestaat daarbij uit een verzameling regels in een regelbank. Verder bestaat een specificatie van een module uit een modulenaam en een signatuur. Een signatuur beschrijft de interface van de module naar de rest van het systeem. Het bevat een declaratie van de objecten en hun typen en de relaties tussen de objecten, die in de module worden gebruikt. Het informatie-schema valt te construeren uit de signatuur door alle toegestane combinaties van objecten en relaties te nemen.

Ad 2.
In essentie is een transformatie een tabel, waarin kan worden afgelezen welke taalelementen van de ene module te maken hebben met welke taalelementen van een andere module. Omdat modulen zich op twee niveaus kunnen bevinden kunnen transformaties drie richtingen hebben. Deze richtingen worden aangeduid met de termen omhoog (van een module op objectniveau naar een module op systeemniveau), omlaag (van een module op systeemniveau naar een module op objectniveau) en neutraal (tussen twee modulen op hetzelfde niveau).

Een specificatie van een transformatie bevat steeds twee elementen:

- Verbindingen (links) tussen namen in de signaturen van de twee betrokken modulen. Deze verbindingen geven aan *welke* namen met elkaar samenhangen.
- Verbindingen die de inhoud van de samenhang tussen de namen beschrijven.

De specificaties van de verschillende soorten transformaties (omhoog, omlaag en neutraal) onderscheiden zich door de typen verbindingen die kunnen worden beschreven. In feite wordt het door de beschrijving van de verbindingen mogelijk om waarheidswaarden vast te stellen van de uitspraken in de output-module van de transformatie, gegeven de inputmodule.

Ad 3.
De controlestruktuur kan worden beschouwd als een netwerk op systeemniveau. De knopen in dat netwerk zijn de modulen van het objectniveau. De kanten van het netwerk bevatten condities, die bepalen of een bepaalde knoop in het netwerk inderdaad bereikbaar is vanuit de module, die op een zeker moment geactiveerd is. Als de geactiveerde module geen conclusies meer kan trekken kan er een andere module worden geactiveerd. Welke module dat is, wordt beschreven in een transitie. De beschrijving van de transities maken deel uit van de controlestruktuur.

3.2.3 VERGELIJKING

Zowel KADS als DESIRE hebben uitgesproken opvattingen over wat een conceptuele specificatie is en welke taalelementen nodig zijn om zo'n specificatie te beschrijven. Een eerste observatie leert, dat KADS zich vooral richt op de kennisanalyse, terwijl DESIRE zich meer richt op de modellering van de kennis ten behoeve van een systeemimplementatie. Anders gezegd: KADS is met name geschikt voor de personen, die zich bezig houden met kennisacquisitie (de knowledge engineers), terwijl DESIRE met name geschikt is voor de personen die zich bezig houden met kennissysteemontwerp (de system engineers). Deze observatie is een verklaring voor de wijze waarop in beide methoden voldaan wordt aan de eerder geformuleerde eisen t.a.v. een conceptuele specificatie en een specificatietaal. Inzicht in de mate, waarin de beide gepresenteerde talen voldoen aan de eisen, die aan een specificatietaal kunnen worden gesteld geeft ook aanwijzingen over in hoeverre een conceptuele specificatie voldoet aan de eisen, die er aan gesteld zijn.

De taalelementen bij KADS zijn de onderscheiden modellen (model of expertise en model of cooperation) en hun deelaspecten (resp. de vier lagen van kennis en hun wijze van formuleren binnen het model of expertise en een zestal deelmodellen binnen het model of cooperation). De taalelementen bij DESIRE zijn de modulen en hun onderliggende struktuur, de transformaties en de controlestruktuur

Tabel 3.1 geeft een vergelijking van de taal van beide methoden en de mate, waarin beide talen voldoen aan de de eerder gestelde eisen:

Eis	Score
Uitdrukkingskracht	KADS hoogstens gelijkwaardig aan DESIRE
Eenduidigheid	KADS minder dan DESIRE
Structurering	KADS gelijkwaardig aan DESIRE
Onzekerheid/onvolledigheid	KADS gelijkwaardig aan DESIRE
Formeel/Niet-formeel	KADS minder dan DESIRE (formeel) KADS beter dan DESIRE (niet-formeel)

Tabel 3.1. Vergelijking KADS en DESIRE

Ter toelichting op tabel 3.1 kunnen de volgende opmerkingen worden gemaakt:

- Een studie in (Schmidt en Oskamp, 1991) heeft aanwijzingen gegeven, dat een specificatie, die met behulp van KADS is gemaakt, om te zetten is naar een specificatie, die met behulp van DESIRE is gemaakt. Om deze reden wordt de uitdrukkingskracht van de specificatietaal van KADS als ten hoogste gelijkwaardig aan de specificatietaal van DESIRE beschouwd.
- DESIRE is gebaseerd op een model met een wiskundige grondslag. Iedere component, die binnen DESIRE wordt onderscheiden, is beschreven met behulp van een wiskundig formalisme, waardoor de betekenis van zo'n component eenduidig is vastgelegd. KADS moet zo'n formalisme ontberen. Dit is de reden, dat DESIRE op Eenduidigheid hoger "scoort" dan KADS. Er bestaan overigens wel aanzetten tot het meer formaliseren van modellen, die binnen KADS worden gebruikt ((Akkermans et al, 1989), (De Hoog, 1992)).
- De taalelementen van DESIRE geven aanknopingspunten tot een hiërarchische opbouw van de specificatie, waarbij ieder lager niveau een verfijning is van ieder hoger niveau. De taalelementen van KADS geven een structurering in de kennis aan, die niet hiërarchisch is in de zin, dat een onderliggend niveau een verfijning is van een bovenliggend niveau. Hoewel bij zowel KADS als DESIRE dus structureringsmogelijkheden aanwezig zijn, zijn de verkregen structuren in de kennis bij de twee methoden sterk verschillend. De "gelijkwaardigheid" in de tabel wat betreft Structurering slaat dan ook alleen op het feit dat structureringsmogelijkheden aanwezig zijn, niet op de wijze van structurering.

- Zowel KADS als DESIRE geven voldoende mogelijkheden om onzekerheid en onvolledigheid te specificeren met behulp van de ter beschikking staande taalelementen. Tevens is het bij beide methoden mogelijk (tijdelijk) onvolledige ontwerpen te specificeren, zonder dat dit ten koste gaat van de eisen die aan een ontwerp kunnen worden gesteld (zoals overdraagbaarheid).
- De resultaten van KADS zijn nog niet geschikt om zonder meer te dienen als input voor een implementatiefase. Zoals ook al opgemerkt bij de bespreking van de methode DESIRE moeten er nog veel aannamen gedaan worden c.q. zaken worden uitgezocht, die voor een goede implementatie van belang zijn (zoals een dialoogstructuur). De specificatie die door DESIRE wordt opgeleverd is qua vorm en inhoud beter geschikt om te dienen als basis voor een systeemimplementatie, maar minder geschikt om te dienen als communicatiemiddel met de gebruiker van het te ontwikkelen systeem. Om die reden "scoort" DESIRE minder dan KADS op het weergeven van niet-formaliseerbare aspecten en geeft het een beter resultaat voor het weergeven van formaliseerbare aspecten.
- Opvallend is, dat noch KADS noch DESIRE bij de conceptuele specificatie aandacht schenkt aan de prestatie-eisen, zoals bij de ANSI/IEEE-standaard is gespecificeerd.

3.3 HET XYFAR-MODEL ALS SPECIFICATIETAAL

Het XYFAR-model, dat in hoofdstuk 2 werd gepresenteerd, biedt een aantal taalelementen, waarmee het in theorie mogelijk is een conceptuele specificatie te maken van een kennissysteem. In deze paragraaf zal deze theorie nader worden onderzocht. Achtereenvolgens zal worden beschreven, hoe een conceptuele specificatie met behulp van de taalelementen uit het XYFAR-model eruit ziet. Dit zal worden gelegd naast de eisen die aan een conceptuele specificatie zijn te stellen. Tenslotte zal een vergelijking worden gemaakt met de conceptuele specificatietaal van DESIRE.

Een specificatie van een kennissysteem volgens het XYFAR-model bestaat uit:

1. Een specificatie van een database-universum
2. Een specificatie van een regelbank
3. Een specificatie van een inferentieprocedure. Deze maakt gebruik van:
   a. Een specificatie van een hypothese-selectiefunctie
   b. Een specificatie van een regel-selectiefunctie
   c. Een specificatie van een externe regel

Een nadere beschouwing van de methode DESIRE leert, dat de daar geformuleerde denkwijze over de werking van een kennissysteem in grote mate overeenkomt met die, die achter het XYFAR-model ligt. Beide aanpakken gaan uit van een database, waar met behulp van regels nieuwe gegevens aan kunnen worden toegevoegd. Verschillen tussen DESIRE en het XYFAR-model zijn te vinden in de uitwerking van de werkwijzen:

- DESIRE beschouwt een kennissysteem expliciet als opgebouwd uit een aantal samenwerkende modulen. Ieder van die modulen heeft een afgebakende taak binnen het totaal. Modulen kunnen echter wel qua functionaliteit gelijkwaardig zijn. Zo kunnen twee modulen beide een redeneertaak hebben, zij het op verschillende domeinen. Gelijkwaardige modulen hebben wel dezelfde opbouw. Het XYFAR-model beschouwt een kennissysteem als opgebouwd uit een aantal onafhankelijk te beschouwen componenten. Ieder van die componenten heeft een bepaalde taak binnen het totaal. Componenten zijn echter niet gelijkwaardig. De eerder beschreven mogelijkheden tot het aanbrengen van een hiërarchie in de kennis, die met DESIRE mogelijk is, ontbreken bij de componenten van het XYFAR-model.
- DESIRE kent geen beperkingen t.a.v. het regelformaat. Het XYFAR-model gaat expliciet uit van Horn-clauses.
- Een specificatie met behulp van het XYFAR-model lijkt dichter tegen een systeemrealisatie te liggen dan die van DESIRE. In het bijzonder blijkt dit bij het vergelijken van de geschiktheid van beide methoden voor het beschrijven van te gebruiken software voor een systeemrealisatie. De taalelementen, die in het XYFAR-model worden gebruikt, geven directere aanknopingspunten voor het maken van zo'n beschrijving dan de wat abstractere taal, die door DESIRE wordt gebruikt. Door software in dezelfde termen te beschrijven als het te ontwikkelen systeem, kan eenvoudig worden bepaald, in hoeverre de beschreven software geschikt is om te dienen als implementatiemiddel voor het systeem. Eventuele afwijkingen in de conceptuele specificatie en een te maken implementatiemodel zijn dan direct duidelijk. In hoofdstuk 4 zal hiervan een voorbeeld worden gegeven. In feite maakt de beschrijving van de software in termen van het XYFAR-model deel uit van het implementatiemodel.
- Doordat een redeneerstrategie bij het XYFAR-model "uiteengerafeld" wordt in diverse systeemcomponenten kan het voorkomen, dat in de uiteindelijke implementatie de strategie niet makkelijk is terug te vinden. Dit kan zich uiten in onduidelijkheden over de aanwezigheid van bepaalde hypothese- of regelselectiefuncties. Een adequate documentatie van iedere (meta)regel kan dit nadeel echter opheffen.

De gebruikte taalelementen kunnen worden getoetst aan de eisen voor een specificatietaal, die eerder in dit hoofdstuk zijn geformuleerd. Vergeleken met DESIRE ontstaat het beeld, zoals in tabel 3.2 is weergegeven.

Eis	Score
Uitdrukkingskracht	XYFAR-model minder dan DESIRE
Eenduidigheid	XYFAR-model gelijkwaardig aan DESIRE
Structurering	XYFAR-model gelijkwaardig aan DESIRE XYFAR-model minder dan DESIRE (hiërarchie)
Onzekerheid/onvolledigheid	XYFAR-model minder dan DESIRE
Formeel/Niet-formeel	XYFAR-model gelijkwaardig aan DESIRE

Tabel 3.2. Vergelijking DESIRE - XYFAR-model

Ter toelichting op tabel 3.2 kunnen de volgende opmerkingen worden gemaakt:

- Het specificeren van een aantal aspecten van een kennissysteem zal met DESIRE wat makkelijker gaan dan met het XYFAR-model. In het bijzonder de minder stringente eisen aan het regelformaat en de gedetailleerder uitwerking van het specificeren van onzekerheid en onvolledigheid geven DESIRE een pré boven het XYFAR-model.
- De gelijkwaardigheid wat betreft Structurering zegt alleen iets over de mogelijkheid tot het aanbrengen van een structuur bij beide talen. Zoals eerder al werd opgemerkt levert DESIRE echter de mogelijkheid tot het geven van een hiërarchie in de kennis. Dit is echter niet mogelijk bij de componenten van het XYFAR-model. Dit verklaart het "minder" zijn van het XYFAR-model t.o.v. DESIRE op het gebied van Structurering.
- Zowel de taal van het XYFAR-model als de taal van DESIRE lenen zich minder voor het specificeren van niet-formele aspecten van een kennissysteem.

Gegeven deze observaties kan worden geconcludeerd, dat de specificatietaal van DESIRE meer mogelijkheden geeft voor het specificeren van een kennissysteem. Het XYFAR-model levert echter wel een specificatie op die beter geschikt is om te dienen als basis voor een computer-implementatie. Dit vanwege het meer instrumentele karakter van de specificatie: er

wordt gespecificeerd in componenten die direct aanwijsbaar zijn in software, geschikt voor de implementatie van kennissystemen. Deze vertaalslag moet voor een specificatie in DESIRE nog worden gemaakt.

Het is tevens duidelijk dat beide specificaties onvolledig zijn vanwege het ontbreken van niet-formele aspecten van een kennissysteem. Hiervoor zijn echter meer traditionele methoden voor systeemontwikkeling te gebruiken. Tevens gaan DESIRE en het XYFAR-model ervan uit dat de kennis reeds achterhaald is. KADS lijkt echter veel geschikter voor juist het achterhalen van de kennis die uiteindelijk de basis van een kennissysteem moet vormen. Vanuit deze observaties kan het vermoeden worden geformuleerd dat het beschrijven van implementatiesoftware met behulp van de specificatietalen van KADS en DESIRE wel mogelijk is, maar meer moeite zal kosten dan met behulp van de taalelementen van het XYFAR-model.

Een werkwijze om te komen tot een conceptuele specificatie van een kennissysteem zou daarom de volgende kunnen zijn:

> Gebruik KADS om te komen tot een eerste conceptuele specificatie. Vertaal deze specificatie ofwel rechtstreeks in de terminologie van het XYFAR-model ofwel via een specificatie met behulp van de taal van DESIRE in een specificatie in de taal van het XYFAR-model.

De keuze, welke van de beide werkwijzen te volgen, kan onder meer afhangen van de gewenste mogelijkheden voor het weergeven van onzekere en onvolledige informatie. Hiervoor biedt DESIRE meer mogelijkheden. Zoals al eerder is vermeld, staat in (Schmidt en Oskamp, 1991) aangegeven, hoe een vertaling van KADS naar DESIRE kan plaatsvinden. In dit kader zal niet worden stilgestaan bij de wijze waarop een vertaling van DESIRE naar het XYFAR-model moet gebeuren.

4 EEN KENNISVOORBEELD: BUREAU-EILAND

Ter illustratie van de theorie uit de voorgaande hoofdstukken zal een voorbeeld worden uitgewerkt. Het voorbeeld betreft de specificatie van een kennissysteem, dat ondersteuning biedt bij het configureren van een een bureau-eiland. De beschrijving is geïnspireerd door (Dumoulin, 1989). Een bureau-eiland betreft een complex samengesteld produkt, waarbij sprake is van een groot aantal beperkingen. De ondersteuning kan bijvoorbeeld verlangd worden door een verkoper, die in dialoog met een klant het produkt configureert naar de wensen van de klant. Voorbeelden van deze problematiek zijn o.a. beschreven in (Euwe en Schuwer, 1992) en (Erens, 1990). Een bekend systeem, dat ondersteuning geeft bij configuratie (van VAX-computers), is XCON (Barker en O'Connor, 1989).

Er zal eerst een informele beschrijving van het probleem worden gegeven. Hierna zal een ontwerp van een kennissysteem worden gepresenteerd in de terminologie van het XYFAR-model. Vervolgens zullen twee implementaties worden beschreven van het kennissysteem, waarna tenslotte beide implementaties met elkaar worden vergeleken.

4.1 PROBLEEMBESCHRIJVING

Door een firma wordt een scala aan kantoormeubilair geleverd. Eén van de te leveren produkten betreft een bureau-eiland. Onder een bureau-eiland wordt verstaan 1 tot 4 bureaus, waarbij in het geval van 2, 3 of 4 bureaus deze middels koppelelementen aan elkaar zijn gekoppeld tot een geheel. Van een bureau moeten materiaal, lengte, breedte, hoogte en kleur worden gespecificeerd. Van een bureau-eiland moet eveneens het aantal bureaus, waarmee het eiland wordt gevormd, bekend zijn. Bureaus kunnen worden voorzien van een aanhangtafel en van ladenblokken. Indien voor een van deze opties wordt gekozen, moeten lengte, breedte en hoogte bekend zijn. Van een ladenblok moet tevens de uitvoering (hang- of legladen) gegeven zijn. Bij bureau-eilanden kunnen bijpassende bureaustoelen worden geleverd. Van een bureaustoel moeten kleur, wel/geen armleuning, wel/niet draaibaar en wel/niet rijdbaar bekend zijn. Totaal bestaan er zo'n 300.000 varianten van dit produkt.

Neem aan dat de produktinformatie opgeslagen wordt in een kennissysteem. Met behulp van dit systeem moet het mogelijk zijn om correcte produktspecificaties te genereren. Deze kunnen als invoer voor een stuklijst- en produktiesysteem worden gebruikt.

Bij het configureren van een bureau-eiland geldt een groot aantal constraints. In de nu volgende beschrijving van de constraints krijgt iedere constraint een aanduiding. Deze

aanduiding zal worden genoteerd als [Rxx], waarbij xx een volgnummer is. Bij de formele beschrijving van het systeem in bijlage B zal naar deze aanduidingen worden terugverwezen.

Een bureau-eiland bestaat uit 1 tot 4 bureaus [R01]. De bureaus worden gefabriceerd uit houten platen of spaanplaten. In het eerste geval is de kleur van het bureau altijd naturel. In het laatste geval wordt het spaanplaat overtrokken met een kunststof laminaat in de kleuren lichtgrijs, grijs, zwart en bruin [R02]. Alle delen uit een bureau-eiland worden gefabriceerd uit hetzelfde materiaal [R03]. De bureaus worden in vier standaardmaten geleverd (l*b*h): (180*80*90),(180*90*90), (200*80*90) en (200*90*90) [R04]. Bureaus in een eiland hebben dezelfde afmetingen [R05].

Een bureau kan met 0,1 of 2 ladenblokken worden geleverd [R06]. Als het bureau met 1 ladenblok wordt geleverd, dan kan dit blok zowel links als rechts worden bevestigd [R07]. Een ladenblok kan in twee uitvoeringen worden geleverd [R08]. Een uitvoering bevat een pennenlade en twee gewone lades. De andere uitvoering bevat een pennenlade en een hangmaplade.

Bureaus in een eiland worden tegen elkaar geschoven met behulp van een koppelsegment. Er bestaan drie soorten koppelsegmenten, een kwart-cirkel, gelijkzijdige driehoek of vierkant, resp. geschikt voor eilanden met 2, 3 of 4 bureaus [R09]. De lengte van de rechte zijde van de kwart-cirkel resp. de lengte van een zijde van een driehoek of een vierkant is gelijk aan de breedte van het bureaublad [R10]. De kleur van een koppelsegment is dezelfde als die van het bureau, maar rood en blauw zijn eveneens toegestane kleuren [R11]. De toegestane kleurcombinaties zijn [R12]:

Kleur bureau	Kleur koppeldeel
lichtgrijs	lichtgrijs, zwart
grijs	grijs, zwart, rood, blauw
zwart	lichtgrijs, grijs, zwart, rood, blauw
bruin	bruin

Alle overige delen van een eiland hebben dezelfde kleur als het bureau [R13]. Door drie kwart-cirkels aan 1 bureau te bevestigen ontstaat een bijzonder geval van een eiland, namelijk een zgn. P-opstelling. Bij een P-opstelling moeten de kleuren van het bureau en de kwart-cirkels hetzelfde zijn [R14]. Tevens moet de lengte van het bureau 200 cm zijn [R15]. Andersom worden bureaus met een lengte van 200 cm alleen in P-opstelling geleverd [R16]. Een P-opstelling is het enige eiland, dat uit 1 bureau bestaat. Losse bureaus kunnen dus niet worden gekocht. [R17]

Bureaus kunnen worden voorzien van een aanhangtafel. De tafel kan zowel links als rechts worden bevestigd. Een links-bevestigde aanhangtafel is qua uitvoering het spiegelbeeld van een rechts-bevestigde aanhangtafel [R18]. Een aanhangtafel kan met of zonder ladenblok worden geleverd [R19]. Een ladenblok wordt dan bevestigd aan die zijde van de aanhangtafel, die niet "aanhangt" tegen het bureau [R20]. Indien een bureau een aanhangtafel heeft, kan aan het bureau maximaal 1 ladenblok worden bevestigd en wel aan de kant, waar de aanhangtafel niet is bevestigd [R21]. In een eiland worden of alle bureaus met of alle bureaus zonder aanhangtafel geleverd [R22]. Bovendien kunnen de aanhangtafels in een eiland met meerdere bureaus niet willekeurig worden bevestigd. De toegestane combinaties zijn [R23]:

Aantal bureaus	Combinaties
2	(L,L), (L,R), (R,R)
3	(L,L,L), (R,R,R)
4	(L,L,L,L)

Bij een bureau kunnen bijpassende bureaustoelen worden geleverd. Stoelen zijn leverbaar in de kleuren rood, grijs, blauw, bruin en groen [R24]. Per bureau wordt een stoel geleverd [R25]. Een bureaustoel is altijd rijdbaar en draaibaar [R26]. Alleen bij een P-opstelling kunnen drie stoelen worden geleverd, die niet rijdbaar en niet draaibaar zijn [R27]. Indien bij een eiland meerdere stoelen worden geleverd, hebben ze alle dezelfde kleur [R28]. De volgende kleurcombinaties van bureaus en stoelen zijn toegestaan [R29]:

Kleur bureau	Kleur stoel
naturel	rood, grijs, blauw, bruin, groen
lichtgrijs	rood, blauw, groen
grijs	rood, blauw, groen
zwart	grijs, bruin
bruin	grijs, bruin, groen

Een stoel kan worden geleverd met of zonder armleuningen [R30].

Bij het maken van een configuratie moeten keuzen worden gemaakt. Een te kiezen element wordt in het vervolg een *parameter* genoemd. Een voorbeeld van een parameter is de kleur van een bureau.

Er zijn verschillende strategieën mogelijk om een configuratie te verkrijgen:

1. Bij het controleren van een configuratie zijn twee typen controles nodig:
   a. Ieder gewenst onderdeel voldoet aan alle randvoorwaarden, die voor dat onderdeel gelden.
   b. De combinatie van gewenste onderdelen (de configuratie) voldoet aan alle randvoorwaarden, die gelden voor die combinatie van onderdelen.

   Het is beter eerst stap a. en dan stap b. te doen. Het is namelijk eenvoudiger om een configuratie samen te stellen, die uitgaat van reeds "goedgekeurde" onderdelen, dan andersom. Daarbij komt dat het veel werk is, om de toelaatbaarheid van een combinatie van onderdelen te testen. Als dan achteraf blijkt, dat een onderdeel van de combinatie niet is toegelaten, dan is dat vele werk voor niets gedaan.
2. Er wordt vaak vanuit een voorbeeld geconfigureerd. Dit is bijvoorbeeld het geval indien een klant reeds eerder een bureau-eiland heeft besteld en later enkele wijzigingen wil aanbrengen in de bestelde configuratie (ervan uitgaande dat de "oude" configuratie nog steeds een toegelaten configuratie is). Ook komt deze situatie voor bij klanten, die nog geen idee hebben van alle mogelijkheden. Er wordt dan begonnen met een soort "default-configuratie", waarin een klant dan wijzigingen kan aanbrengen. Tenslotte kan een variant van deze laatste strategie worden gebruikt. Bij deze variant worden eerst enkele parameters gekozen, waarna voor de rest van de parameters een default waarde wordt genomen.
3. Om de verzameling mogelijke configuraties zo snel mogelijk in te perken, moet op ieder moment de dan meest discriminerende parameter het eerst worden gekozen. De meest discriminerende parameter is die parameter, die de verzameling nog toegelaten configuraties het meest verkleint.
4. Bij de configuratie heeft een klant vaak voorkeuren voor de volgorde, waarin de parameters worden gekozen. Zo kan het voorkomen, dat de klant eerst de kleur van het koppeldeel bepaalt, waarna hij de rest van de parameterwaarden bepaalt.

Merk op, dat strategie 4 in strijd kan zijn met strategie 3. De voorkeursvolgorde van een klant hoeft niet per se de optimale volgorde te zijn vanuit het oogpunt van configuratie-efficiency bekeken. Evenzo kan strategie 2 met strategie 3 of met strategie 4 worden gecombineerd door bij de bepaling van het al dan niet accoord zijn van de defaultwaarden ófwel de meest efficiënte wijze, ófwel de door de klant gewenste wijze van configureren aan te houden.

Bij het toetsen op toelaatbaarheid van een configuratie blijkt vooral de combinatie van de constraints R07 en R20 moeilijkheden op te leveren. Vaak worden bureau-eilanden besteld, waarbij het ladenblok aan dezelfde kant van het bureau moet worden bevestigd als de aanhangtafel. Ook de gewenste kleurcombinaties van de diverse onderdelen (constraint R12)

leveren vaak niet-toegelaten configuraties op. Tenslotte blijkt, dat meestal een configuratie wordt gewenst, waarin twee ladenblokken en een aanhangtafel voorkomen.

4.2 CONCEPTUEEL ONTWERP

De configuratietaak kan worden gedecomponeerd in de volgende drie deeltaken:

1. Afhankelijk van het type gebruiker wordt een van de strategieën 2, 3 of 4 uit paragraaf 4.1 geselecteerd. (Merk op, dat strategie 1 met zowel strategie 2, 3 als 4 kan worden gecombineerd.)
2. Met behulp van de strategie, die in deeltaak 1 is geselecteerd wordt een configuratie bepaald. Bij het bepalen van de configuratie geeft de gebruiker zijn eisen als invoer. De resulterende (deel)configuratie kan geheel of slechts gedeeltelijk zijn gespecificeerd. In het laatste geval zijn er een of meer parameters, waarvoor het systeem waarden moet bepalen.
3. Het systeem controleert of de (deel)configuratie aan alle constraints voldoet. Het gebruikt hierbij strategie 1. Het resultaat van de controle is "GOED" of "FOUT".

Deeltaken 2 en 3 worden herhaald totdat de gebruiker tevreden is met de dan verkregen resultaten. Als er geen configuratie naar tevredenheid van de gebruiker kan worden bepaald, kan deeltaak 1 worden gebruikt voor het aangeven van een andere strategie.

Er zijn andere wijzen van configuratie mogelijk. Zo zou aan de gebruiker alleen toegestane keuzen kunnen worden getoond, waarbij na iedere keuze wordt aangegeven welke nog te maken keuzen onmogelijk zijn geworden als gevolg van een constraint (Euwe en Schuwer, 1992). De essentie van het controleren van een (deel)configuratie wijkt bij andere configuratiestrategieën echter niet af van de hiervoor genoemde. Het XYFAR-model geeft echter geen ondersteuning bij het vinden van deze decompositie van deeltaken.

Deeltaak 2 en 3 kunnen meer in detail worden beschreven door de volgende procedures, weergegeven in Nassi-Shneidermann diagrammen.

Deeltaak 2: Bepaal configuratie

<table>
<tr><td colspan="4">Bepaal start-configuratie volgens gekozen strategie</td></tr>
<tr><td colspan="4">Zolang configuratie niet accoord</td></tr>
<tr><td colspan="3">Lees(gebruikerswensen)</td><td></td></tr>
<tr><td colspan="3">Parameters onbepaald</td><td></td></tr>
<tr><td colspan="3">ja</td><td>nee</td></tr>
<tr><td colspan="3">Zolang nog parameters onbepaald en niet-onderzochte onderdelen in database (1)</td><td>Controleer configuratie (deeltaak 3)</td></tr>
<tr><td></td><td colspan="2">Zoek in database naar overeenkomstig onderdeel</td><td></td></tr>
<tr><td></td><td colspan="2">Voeg toe aan configuratie</td><td></td></tr>
<tr><td></td><td colspan="2">Controleer configuratie (deeltaak 3)</td><td></td></tr>
<tr><td></td><td colspan="2">Configuratie OK</td><td></td></tr>
<tr><td></td><td>ja</td><td>nee</td><td></td></tr>
<tr><td></td><td>Voeg parameterwaarden toe aan antwoord</td><td>Skip</td><td></td></tr>
<tr><td colspan="4">Schrijf(antwoord)</td></tr>
</table>

(1): met "niet-onderzochte onderdelen" worden die eiland-onderdelen in de database bedoeld, die aan de gebruikerswensen voldoen, maar die nog niet door het systeem zijn onderzocht op toelaatbaarheid in de configuratie.

Deeltaak 3: Controleer configuratie

Lees(configuratie)			1
Zolang onderdeel niet-gecontroleerd en alle gecontroleerde onderdelen accoord			2
	Selecteer onderdeel		3
	Controleer onderdeel		4
Alle onderdelen accoord			5
ja		nee	
Controleer configuratie		Skip	6
Schrijf(resultaat)			7

Voor ieder van de deeltaken 2 en 3 kan een kennissysteem worden ontworpen, waarbij deeltaak 3 dan een implementatie is van het controle-mechanisme CM. Bij dit configuratieprobleem is het echter juist het controle mechanisme, dat het probleem niet-triviaal maakt. In het navolgende wordt een ontwerp van een kennissysteem gegeven als een mogelijke implementatie voor dit controle mechanisme (in de terminologie van hoofdstuk 2). De query, die door deeltaak 2 aan dit te ontwerpen systeem wordt aangeboden, kan de volgende vormen hebben:

1. Het geval dat er geen volledige configuratie van een bureau-eiland wordt gevraagd:
   Is de combinatie van bureau B en koppeldeel K en ... een toegelaten combinatie?
2. Het geval dat er een volledige configuratie van een bureau-eiland wordt gevraagd:
   Is de configuratie, die bestaat uit bureau B en koppeldeel K en ... een toegelaten configuratie?

Er wordt dus aangenomen, dat tenminste een bureau en een koppeldeel zijn aangegeven (al dan niet volledig gespecificeerd door de gebruiker). Voor alle onderdelen, die niet zijn aangegeven, wordt door deeltaak 2 een code 0 (niet gekozen cq niet gespecificeerd) ingevuld. Het is overigens eenvoudig, om deze aanname te laten vervallen. Aan de regelbank zullen dan regels moeten worden toegevoegd voor die gevallen, waarbij het koppeldeel en/of het bureau niet zijn aangegeven (cq een code 0 hebben). Tevens zal aan het waardenbereik van

de attributen Bureaucode cq Koppelcode de waarde 0 moeten worden toegevoegd. Tenslotte zal een regel moeten worden toegevoegd die aangeeft, wanneer een combinatie van onderdelen toegelaten is, als geen sprake is van een volledige configuratie. Als dit niet gebeurt, zal een niet-complete configuratie toch accoord kunnen worden bevonden (bedenk dat een bureau-eiland tenminste een bureau en een koppeldeel bevat).

4.2.1 DATABASE

Een formele definitie van het DB-universum is te vinden in bijlage B. Uit deze definitie is af te lezen, dat er twee soorten entiteittypen zijn te onderscheiden:

1. Entiteittypen voor de representatie van de eiland-onderdelen en de configuraties
2. Entiteittypen voor de representatie van de toegestane combinaties van onderdelen in een configuratie.

Deze database-definitie kan worden herschreven in de stijl van hoofdstuk 2, door in plaats van iedere tabelnaam een predicaat te definiëren met het aantal attributen als ariteit. De plaats van de argumenten binnen het predicaat bepaalt de betekenis, die aan dat argument moet worden toegekend.

VOORBEELD
Definieer het relatiesymbool BUREAU met ariteit 5. Het eerste argument is de bureaucode, het tweede argument is het materiaal etc.
□

Een aantal beperkingen, die hiervoor zijn genoemd, vallen niet binnen de scope van dit systeem: [R06], [R09] en [R18] zijn gegevens, die pas van belang zijn voor het genereren van de stuklijst.

Bij de formulering van de database worden ook tupels toegestaan met een sleutelwaarde 0. Deze representeren de situatie, dat het betreffende item niet wordt gekozen. Dit geldt voor de optionele items ladenblok, aanhangtafel en stoel.

4.2.2 DE REGELBANK

Met de regels in de regelbank moet het mogelijk zijn af te leiden of aan de constraints is voldaan. In feite is, bij deze implementatie van het controle mechanisme, de regelbank de kern.

VOORBEELD
De regels, die de implementatie vormen van constraint R15:

ALS het bureau een lengte heeft van 200 cm
EN het koppeldeel is bestemd voor een eiland met 1 bureau
DAN is er voldaan aan constraint R15.

ALS het bureau een lengte heeft ongelijk aan 200 cm
EN het koppeldeel is bestemd voor een eiland met meer dan 1 bureau
DAN is er voldaan aan constraint R15.
□

De aangeboden configuratie zal een combinatie van eiland-onderdelen zijn. Het systeem controleert of de configuratie, die door die combinatie van eiland-onderdelen ontstaat, toegevoegd aan de bestaande database een toegelaten database oplevert. Er is daarom ook een regel, die een configuratie berekent:

ALS B een bureau
EN K een koppeldeel
EN A een aanhangtafel
EN L1 een ladenblok
EN L2 een ladenblok
EN S1 een stoel
EN S2 een stoel
DAN {B,K,A,L1,L2,S1,S2} is een configuratie

Tevens is er een regel, die aangeeft, wanneer een configuratie een toegelaten configuratie is:

ALS X een configuratie is
EN X voldoet aan constraint R03a
EN X voldoet aan constraint R03b
EN ...
DAN is X toegelaten

Deze laatste regel maakt gebruik van de twee eerdergenoemde typen regels. Er ontstaat zo een natuurlijke hiërarchie tussen de regels in de regelbank.

4.2.3 HYPOTHESE-SELECTIEFUNCTIES

De eerder genoemde strategie 1 (eerst het toegelaten zijn van de eiland-onderdelen nagaan en dan pas het toegelaten zijn van de combinatie) kan worden geïmplementeerd met behulp van de volgende hypothese-selectiefunctie:

> ALS in de query de hypothese CONFIG(,,,,,,,) voorkomt
> DAN moet deze hypothese als laatste worden geselecteerd

Voor de andere hypothesen kan de volgende hypothese-selectiefunctie worden geformuleerd:

> Selecteer de hypothesen in de tekstuele volgorde

4.2.4 REGEL-SELECTIEFUNCTIES

De kennis over de eerder geschetste veel voorkomende problemen bij het configureren van een eiland (het regelmatig schenden van constraints R07 en R12) en de observatie, dat een configuratie met twee ladenblokken en een aanhangtafel het meest wordt gewenst, kan worden gebruikt bij de formulering van de volgende regel-selectiefunctie:

Selecteer de regels in de volgorde:
> (Indien van toepassing) eerst regels met R07 in de kop
> > Daarbij eerst de regel, die geldt voor de combinatie met twee ladenblokken en een aanhangtafel
> (Indien van toepassing) vervolgens de regels met R12 in de kop
> Vervolgens de overige regels in de tekstuele volgorde

Indien een van de constraints wordt geschonden, stopt het systeem met afleiden en geeft het antwoord "FALSE" aan de gebruiker (i.c. deeltaak 2). Dit is in overeenstemming met de werking, die in het XYFAR-model van hoofdstuk 2 werd geschetst. Om deze reden geeft de hier geformuleerde regel-selectiefunctie veelal een efficiënt afleidingsproces. De constraints die het meest worden geschonden worden immers als eerste nagegaan.

4.2.5 EXTERNE REGEL

Het ligt voor de hand om bij dit systeem te kiezen voor de CWA als externe regel. Als van een hypothese niet kan worden nagegaan of het element is van de deductieve afsluiting van de database onder de regelbank, dan zal een eiland-onderdeel zijn gespecificeerd, dat niet in de database is terug te vinden (bijvoorbeeld een bureau met een paarse kleur). Configuraties met niet-bestaande onderdelen zijn natuurlijk niet toegelaten.

4.2.6 UITGEBREIDE REDENEERPROCEDURE

Het eerder gegeven Nassi-Shneidermann diagram van deeltaak 3 geeft tevens de structuur van de redeneerprocedure weer. Een verfijning in deze procedure is:

Regel 6:
Controleer configuratie met behulp van de backward chaining strategie.

De keuze voor de backward chaining strategie is genomen vanwege de hiërarchie in de regels, die bij de beschrijving van de regelbank werd aangegeven.

4.3 TWEE IMPLEMENTATIES VAN HET BUREAU-EILAND CONFIGURATIESYSTEEM

In hoofdstuk 3 is opgemerkt, dat het XYFAR-model kan dienen als een middel om de meest geschikte software voor een implementatie van een te realiseren systeem te bepalen. Hiertoe kunnen de specificaties van het systeem en die van de software worden opgesteld in termen van het XYFAR-model. Door beide specificaties met elkaar te vergelijken, kan worden bepaald of een gegeven softwaretool geschikt is als implementatiemiddel. "Geschikt" betekent in deze context dat de conceptuele specificatie met zo min mogelijk aanpassingen

gerealiseerd kan worden met behulp van de softwaretool. Om te bepalen of software "geschikt" is, kan per component uit het XYFAR-model worden aangegeven, hoe deze is ingevuld in de betreffende softwaretool. Eveneens kan worden aangegeven, of die component door de gebruiker te beïnvloeden is (bijvoorbeeld door aanpassing van die component).

Los van dit geschiktheidscriterium kunnen ook andere criteria een rol spelen bij de selectie van de meestgeschikte tool. Voorbeelden van zulke criteria zijn: budget, hardware-platform (is de software te gebruiken op de aanwezige hardware), context van het systeem (is het een proefsysteem of moet het daadwerkelijk worden gebruikt in een produktie-omgeving? Indien het laatste, moeten er koppelingen mogelijk zijn met reeds aanwezige informatiesystemen en databases?), etc.

Om deze mogelijkheden van het XYFAR-model te illustreren, is voor twee softwaretools (Turbo-Prolog en KES-II) een beschrijving gemaakt met behulp van het XYFAR-model. De tool Turbo-Prolog is een implementatie van de taal Prolog. De tool KES-II is een zgn. shell. Zoals in hoofdstuk 1 al werd beschreven is een shell een kennissysteem, waarbij de domeinafhankelijke aspecten nog niet zijn ingevuld. Meestal betekent dit, dat er een redeneerprocedure aanwezig is, waarbij nog een database en een regelbank moeten worden gedefinieerd. Om deze reden wordt een shell ook wel een "leeg kennissysteem" genoemd. Achtereervolgens zullen eerst de beide tools worden besproken en zal een specificatie van beide tools worden gegeven in termen van het XYFAR-model. Vervolgens zullen de implementaties van het bureau-eiland-probleem worden vergeleken. Onder meer zal worden besproken, welke implementatie-gevolgen de specifieke tool-eigenschappen hebben (bijvoorbeeld de gevolgen voor de implementatie van de eerder onderkende redeneerstrategieën).

4.3.1 TURBO-PROLOG

De taal Prolog ontstond in de jaren 70 en werd al snel beschouwd als een alternatief voor de taal LISP voor implementaties van AI-applicaties. De taal ontstond vanuit het idee, dat logica gebruikt kon worden als een programmeertaal (Lloyd, 1987). Dit idee ontstond door de observatie dat een implicatie in logica (d.i. een formule van de vorm $A \rightarrow B$) ook een procedurele interpretatie kon hebben. In dat geval kan een implicatie worden opgevat als de implementatie van een procedure-definitie. Als aan een verzameling implicaties een vraag wordt gesteld van de vorm $\leftarrow C_1,C_2,..,C_k$, dan kan ieder van de C_i's worden opgevat als een procedure-aanroep. Het bepalen, welke van de implicaties (procedures!) dan worden aangeroepen, gebeurt door de C_i te vergelijken met de kop van iedere implicatie (unificatie).

In "zuiver" Prolog worden slechts Horn-clauses gebruikt voor het declaratief weergeven van het programma. Na het opgeven van een query aan het systeem zorgt het afleidings-mechanisme van Prolog ervoor, dat de query wordt beantwoord door na te gaan of de hypothesen uit de query kunnen worden afgeleid vanuit de weergegeven clauses uit het programma. Het afleidingsmechanisme kan worden gekarakteriseerd als een depth-first zoekalgoritme, waarbij gebruik wordt gemaakt van backtracking. Dit afleidingsmechanisme is bewijsbaar juist en volledig voor een verzameling Horn-clauses. De "zuiverheid" van het zo gebruiken van Prolog komt dan voort uit de realisatie van declaratief programmeren (WAT), zonder rekening te hoeven houden met het HOE.

Prolog kent echter faciliteiten, waardoor deze zuiverheid deels teniet wordt gedaan. De meest "vervuilende" operator daarbij is de "!" (de cut-operator). Deze operator kan aan clauses worden toegevoegd en heeft de procedurele semantiek, dat op de plaats van deze operator backtracking niet meer mogelijk wordt. Dit geeft de programmeur de mogelijkheid het redeneermechanisme te beïnvloeden, maar maakt tegelijk de leesbaarheid van het programma een stuk lastiger. Tevens treedt er op die wijze een vermenging op van redenering en kennis. Omdat dit ingaat tegen de ideeën die achter het XYFAR-model liggen, is bij de implementatie in Turbo-Prolog geen gebruik gemaakt van deze operator.

Turbo-Prolog wijkt in bepaalde opzichten af van een Prolog-standaard, zoals die bijvoorbeeld te vinden is in (Clocksin en Mellish, 1987). De in dat boek beschreven standaard komt overeen met wat wel de Edinburgh-standaard voor Prolog wordt genoemd. De belangrijkste afwijkingen van deze standaard voor Turbo-Prolog zijn:

- In Turbo-Prolog is het verplicht aan het begin van het programma de gebruikte predikaatsymbolen te definiëren.
- Turbo-Prolog kent diverse ingebouwde mogelijkheden die bij de standaard niet zijn beschreven, zoals database-definitiemogelijkheden en diverse ingebouwde functies.
- Bij Turbo-Prolog bestaat er de verplichting om regels met dezelfde kop te groeperen. Deze verplichting bestaat niet bij standaard Prolog.

De componenten, die in het XYFAR-model worden onderscheiden zijn op de volgende wijze in Turbo-Prolog terug te vinden:

DATABASEDEEL

Turbo-Prolog maakt geen onderscheid tussen gegevens en regels. Een gegeven wordt beschouwd als een regel met een lege staart. Om performance-redenen moeten regels (en feiten) met dezelfde kop bij elkaar worden geplaatst. Turbo-Prolog kent slechts een minimaal constraint-mechanisme: alleen de typering van de gegevens (alfanumeriek of numeriek) geeft een automatische controle op het naleven van de constraints. Alle andere constraints moeten worden geprogrammeerd.

REGELBANK

De regels bij Turbo-Prolog worden weergegeven in het formaat, zoals onderscheiden in het XYFAR-model (Horn-clauses). Standaard is er geen mogelijkheid tot het weergeven van onzekerheid. Deze mogelijkheid kan echter wel worden geprogrammeerd.

REDENEERDEEL

De *regel-selectiefunctie* kan worden omschreven als selectie van de in aanmerking komende regels volgens de tekstuele volgorde. De *hypothese-selectiefunctie* kan eveneens worden omschreven als selectie van de hypothesen in de volgorde, waarin ze worden opgegeven. De *externe regel* is de Negation as Finite Failure regel (NFF-regel). Deze regel is voor positieve hypothesen analoog aan de Closed World Assumption. Indien een negatieve hypothese moet worden bewezen, probeert Prolog volgens de NFF-regel de tegengestelde positieve hypothese in een eindig aantal redeneerstappen te weerleggen. Indien dit lukt, wordt de negatieve hypothese als TRUE beschouwd. Indien de tegengestelde positieve hypothese wordt bewezen, wordt de negatieve hypothese als FALSE beschouwd. Indien het niet in een eindig aantal stappen lukt, dan wordt theoretisch de waarde FALSE aan de negatieve hypothese gegeven. In de praktijk zal het programma echter geforceerd afbreken met een foutmelding. De *redeneerprocedure* gebruikt een backward-chaining strategie voor het redeneren. Hierbij worden alle redeneerregels gebruikt, die in hoofdstuk 2 zijn onderscheiden (inclusief de redeneerregel voor het non-monotoon redeneren). Zoals al eerder opgemerkt, is het binnen Turbo-Prolog mogelijk de redeneerwijze te beïnvloeden door het toevoegen van procedurele commando's, zoals de cut-operator.

EVALUATIE

Zoals uit de beschrijving al is op te maken, zijn binnen Turbo-Prolog componenten niet afzonderlijk benaderbaar. De volgende observaties geven ondersteuning aan deze uitspraak:

- Een gegeven wordt synoniem beschouwd aan een regel. Tesamen met de eis, dat regels met dezelfde kop bij elkaar moeten worden geplaatst, geeft dit een mix van database en regelbank.

- Een ander gevolg van deze eis, gecombineerd met de standaard selectiefuncties binnen Prolog, is dat bij het gebruik van recursieve functies de stopconditie vóór de regels moet worden gezet. Indien dit niet gebeurt, dan raakt het redeneerproces in een oneindige recursie-loop. Omdat bij de implementatie van het bureau-eiland geen recursie werd gebruikt, hoefde hier geen rekening mee worden gehouden.
- Bij de formulering van de database en regelbank moet rekening worden gehouden met de regel-selectiefunctie door in een aantal gevallen gegevens te specificeren voorafgaand aan de regels met dezelfde kop als het gegeven.
- Redeneerstrategieën kunnen alleen expliciet worden gemaakt door het toevoegen van de cut-operator aan regels. Dit geeft echter een vermenging van redeneeraspecten en de regelbank. Indien die vermenging niet wordt gebruikt, blijven redeneerstrategieën impliciet, doordat ze "slechts" zijn terug te vinden in de volgorde waarin de regels zijn gespecificeerd danwel in de volgorde, waarin de hypothesen aan het systeem worden aangeboden.

4.3.2 KES-II

KES-II is een shell, geschreven in C. Deze shell geeft de programmeur een grote verzameling ingebouwde functies, zowel voor het specificeren van de database en de regelbank als voor het specificeren van de redeneerprocedure. KES-II kent een aantal standaard datatypen (o.a. integer, real, string, boolean) en het biedt tevens de mogelijkheid zelf datatypen te definiëren. Een KES-II-programma bestaat uit een aantal secties, waarvan een aantal optioneel zijn. Iedere sectie heeft een bepaald doel binnen het geheel. Binnen KES-II is het mogelijk aan attributen defaultwaarden mee te geven. Ook is er de mogelijkheid zgn. *demons* te specificeren. Een demon is een procedure van de vorm: Als conditie Dan actie. Indien de conditie waar wordt (d.i. indien attributen, die in de conditie voorkomen een bepaalde waarde hebben gekregen), wordt de actie uitgevoerd. Een actie kan bijvoorbeeld zijn het aanroepen van een specifieke regel of het weergeven van een boodschap op het scherm, maar ook het sturen van het redeneerproces door bijvoorbeeld een bepaalde hypothese te laten afleiden. Tevens kent een KES-II-programma een Actions-sectie. Hierin wordt door middel van een procedurele taal op een hoog abstractieniveau de acties aangegeven, die het systeem achtereenvolgens moet uitvoeren.

De componenten van het XYFAR-model zijn op de volgende wijze terug te vinden in een KES-II-programma.

DATABASEDEEL

De basisbouwstenen voor het specificeren van de database zijn de attributen (in de attributes-sectie) en de classes (in de class-sectie). De attributes-sectie maakt het mogelijk "losse" attributen te definiëren. In feite is dit overbodig, omdat dezelfde mogelijkheden, die attributen geven, ook haalbaar zijn met het gebruik van alleen classes. Een class binnen een KES-II-programma is te vergelijken met een entiteittype. Het is mogelijk een hiërarchie tussen classes te definiëren. Hierdoor wordt een class deels vergelijkbaar met het Object-oriented class-begrip. Binnen de definities is het mogelijk voor ieder attribuut een default-waarde mee te geven. Tevens kan aan ieder attribuut teksten worden meegegeven, die op het scherm verschijnen indien de waarde van dat attribuut aan de gebruiker moet worden gevraagd. Het is beperkt mogelijk constraints te definiëren. Alleen waardenverzamelingen op attribuutniveau worden automatisch gecontroleerd. Alle andere constraints moeten worden geïmplementeerd met behulp van regels in de regelbank. Met behulp van demons kunnen externe bestanden worden ingelezen, waarin class-occurrences staan.

REGELBANK

De regels worden gespecificeerd in de rules-sectie. Het is mogelijk bij een regel een tekst te specificeren, die op het scherm verschijnt als een gebruiker om uitleg vraagt (How of Why). Het is mogelijk om een onzekerheidsfactor aan een regel te geven. Met behulp van demons kunnen ook attribuutwaarden worden afgeleid.

REDENEERDEEL

De redeneerprocedure wordt geïmplementeerd in de demons-sectie en de actions-sectie. Voor het specificeren van de actions-sectie staat een procedurele 4GL-taal ter beschikking. Deze taal kent diverse kennissysteem-specifieke commando's. Het commando Obtain attribuutwaarde bijvoorbeeld geeft het programma de opdracht de waarde af te leiden van de opgegeven attribuutwaarde, waarbij een backward-chaining strategie wordt gebruikt. Dit commando kan tevens worden beschouwd als een deel van de implementatie van de hypothese-selectiefunctie, namelijk het door de gebruiker te specificeren deel. De standaardwijze van hypothese-selectie is in de volgorde van voorkomen. De standaard regel-selectiefunctie is eveneens in volgorde van voorkomen van regels in de regelbank. Door het juist gebruik van demons is het echter mogelijk de regel-selectiefunctie te beïnvloeden. Eveneens kan door het gebruik van demons de standaard backward-chaining strategie worden beïnvloed. De door het systeem gebruikte externe regel is de Closed World Assumption. Deze regel kan echter worden "overruled" door het specificeren van default waarheidswaarden. Een default waarheidswaarde moet dan worden opgevat als "indien geen bewijs van het tegengestelde bestaat, is de default waarheidswaarde de aan het attribuut te geven waarde." Overigens kunnen ook defaultwaarden voor niet-boolean attributen worden

aangegeven. Dit maakt redeneren mogelijk, waarbij een attribuutwaarde in een databasetoestand wordt overschreven met de waarde, die wordt afgeleid door middel van een redenering. Deze vorm van niet-monotoon redeneren valt echter buiten de scope van het XYFAR-model, omdat bij deze invulling van een externe regel de externe regel al gebruikt kan gaan worden, voordat de deductieve afsluiting is berekend. Deze wijze van redeneren kent zijn eigen moeilijkheden. Zo kan de redenering zijn ontstaan, uitgaande van defaultwaarden voor bepaalde attributen. Indien in een later stadium deze defaultwaarden niet juist blijken te zijn, moet de redenering eigenlijk weer worden opgevat vanaf het punt, waar de aanname van de defaultwaarde werd gedaan.

EVALUATIE

De componenten uit het XYFAR-model zijn over het algemeen aanwijsbaar terug te vinden in een KES-II-programma. Verder treedt alleen vermenging van componenten op bij het specificeren van de constraints. Dit moet namelijk bij KES-II in de regelbank gebeuren. Ook het gebruik van demons kan zorgen voor een vermenging van componenten. Alleen het specificeren van de externe regel door middel van defaultwaarden is een mogelijkheid, die door het XYFAR-model wordt uitgesloten. De mogelijkheden, die in een KES-II-programma aanwezig zijn om redeneercomponenten te specificeren, maken het een flexibel gereedschap voor de implementatie van een kennissysteem. Verder zijn de gebruikte declaratieve taal en de procedurele 4GL gebruiksvriendelijk, zeker in vergelijking met Prolog.

4.3.3 VERGELIJKING VAN DE IMPLEMENTATIES

Nadat de beide tools in de termen van het XYFAR-model zijn beschreven, kan per tool worden nagegaan, hoe de componenten uit de conceptuele specificatie moeten worden geïmplementeerd. In het bijzonder is het interessant te bekijken, op welke wijze de onderkende strategieën in de termen van hypothese-selectie, regel-selectie en redeneerprocedure in de implementaties zijn terug te vinden.

Turbo-Prolog

Een configuratie wordt als goal aangeboden. Ieder bureau-onderdeel, dat deel uitmaakt van de configuratie, wordt aangeduid met behulp van de bijbehorende waarde van het sleutelattribuut. In een onderdelendatabase worden de diverse onderdelen opgezocht, waarna de verschillende beperkingen kunnen worden gecontroleerd. Indien geen complete configuratie wordt aangeboden, maar slechts de combinatiemogelijkheid van een aantal onderdelen moet worden nagetrokken, wordt ieder onderdeel als hypothese aangeboden door

middel van het aangeven van een sleutelwaarde. De strategieën zijn als volgt terug te vinden in de implementatie:

- Indien een complete configuratie wordt aangeboden, staat de configuratie-hypothese als laatste genoemd. Dit is een gevolg van de hypothese-selectieregel binnen Turbo-Prolog (keuze van de hypothesen in de tekstuele volgorde).
- In de regel, die aangeeft, wanneer sprake is van een goede configuratie staan de condities R07 en R12 als eerste genoemd. Dit is ook een gevolg van de hypothese-selectiefunctie en de backward-chaining strategie. Hierdoor wordt bereikt, dat eerst de regels R07 en R12 worden gecontroleerd. Dit wijkt af van de idee in de conceptuele specificatie, waar alleen de regel-selectiefunctie werd genoemd als implementatie van deze strategie. Hier blijkt dus de beïnvloeding, die de diverse componenten binnen Turbo-Prolog op elkaar hebben. Dit wordt veroorzaakt door het eerder opgemerkte feit, dat de strategieën op deze wijze impliciet in het programma zijn vastgelegd (namelijk alleen door volgorden van regels en hypothesen).
- In de verzameling regels, die de implementatie vormen van R07 staat de regel, die betrekking heeft op een configuratie met twee ladenblokken en een aanhangtafel als eerste genoemd. Dit is een gevolg van de regel-selectiefunctie.

Verder kan rekening worden gehouden met de hypothese-selectiefunctie en regel-selectie-functie van Prolog door binnen regels bepaalde testen in te bouwen, die het redeneerproces efficiënter maken. Het volgende voorbeeld geeft een illustratie hiervan.

VOORBEELD

Bij de implementatie van de combinatie van constraints R07 en R20 wordt een aantal gevallen onderscheiden. Eén geval geeft de weergave van deze constraints indien twee ladenblokken en geen aanhangtafel worden gekozen. De beide ladenblokken mogen dan niet aan dezelfde zijde van het bureau worden gemonteerd. De bijbehorende regel in de implementatie in Turbo-Prolog luidt:

ALS er geen aanhangtafel is gekozen (*)
EN een ladenblok L1 aan bureau B vastzit aan zijde LB1
EN L1 is ongelijk aan 0 (*)
EN een ladenblok L2 aan bureau B vastzit aan zijde LB2
EN L2 is ongelijk aan 0 (*)
EN LB1 ongelijk is aan LB2
DAN is er voldaan aan de constraints R07 en R20

Indien Prolog deze regel evalueert (volgens het backward chaining principe) dan worden (volgens de hypothese-selectiefunctie van Prolog) de voorwaarden in de staart van de regel volgens de tekstuele volgorde verwerkt. De testen bij (*) staan, bij deze metaregel, zodanig dat het redeneerproces de meest efficiënte is. Door het tussenvoegen van de testen (*) wordt voorkomen dat met de regel verder wordt geredeneerd, indien de configuratie niet voldoet aan de voorwaarde dat er geen aanhangtafel en twee ladenblokken zijn gekozen. Bij de formulering van deze regel moet dus op deze wijze rekening worden gehouden met de hypothese-selectiefunctie. Andersom is het echter heel moeilijk om, bij de bestudering van deze regel, achter de (impliciete) hypothese-selectiefunctie te komen. Dit kan bij verandering van deze regel leiden tot een minder efficiënt redeneerproces, doordat dit type testen op een mindere positie (uit het oogpunt van efficiency) binnen de staart van de regel komen.
□

KES-II

In de actions-sectie wordt door middel van het commando "Obtain GoedeConfiguratie" de redenering gestart, die moet leiden tot het gewenste antwoord. De configuratie wordt vanuit een externe file ingelezen via een demon. Er is geen verschil in werking tussen het aanbieden van een complete configuratie of een niet-complete configuratie. De strategieën zijn als volgt terug te vinden in de implementatie:

- De strategie, dat een complete configuratie wordt gecontroleerd nadat alle onderdelen zijn gecontroleerd, is expliciet in de implementatie terug te vinden. Dit is gedaan door de definitie van een aantal demons. Deze keuze is een ontwerpkeuze geweest. Het systeem had ook zonder deze demons kunnen worden geprogrammeerd door gebruikmaking van de standaard hypothese-selectiefunctie. Het voordeel van het gebruik van de demons is echter het expliciet maken van de strategie.
- De toegelaten combinaties van onderdelen zijn terug te vinden in de configuratieconstraints. Binnen de configuratieconstraints zijn echter ook regels te vinden, die betrekking hebben op slechts een onderdeel. Deze regels worden als eerste gecontroleerd. Dit wordt bereikt door in de regel, die aangeeft wanneer aan de configuratieconstraints is voldaan, de betreffende attributen aan het begin te zetten. Deze implementaties zijn het gevolg van de standaard hypothese-selectiefunctie (selectie op tekstuele volgorde) en de standaard backward-chaining strategie.
- Door middel van twee demons kan worden bereikt, dat eerst de configuratieconstraints R07 en R12 worden gecontroleerd.

Uit de ervaringen met de implementaties kunnen verschillende conclusies worden getrokken:

- Afgezien van de cut-operator biedt Turbo-Prolog weinig mogelijkheden om het redeneermechanisme te beïnvloeden. Alleen door een zekere volgorde van formuleren van regels en formuleren van de condities binnen een regel of query kan een redeneerstrategie door de gebruiker worden beïnvloed.
- De demons binnen KES-II zijn een krachtig mechanisme voor het beïnvloeden van het redeneermechanisme. Dit heeft tevens als voordeel, dat de redeneerstrategieën expliciet te maken zijn binnen het programma.
- De standaard selectiefuncties (selectie in tekstuele volgorde voor zowel de regels als de hypothesen) hebben tot gevolg, dat door het specificeren van regels in een regelbank strategieën voor het redeneren worden vastgelegd. Dit impliciet laten van strategieën heeft een minder inzichtelijk systeem tot gevolg. Dit kan worden ondervangen door in de systeemdocumentatie deze impliciete vertaling te onderkennen. Omdat deze implementatie van de selectiefuncties bij veel tools voorkomen en het bij veel tools tevens de enige mogelijkheid is om de redeneerstrategie zo te beïnvloeden, is dit een belangrijk punt om te onderkennen bij de specificatie, implementatie en documentatie van een kennissysteem.

Het gebruik van het XYFAR-model als taal voor zowel het beschrijven van de tool als het specificeren van het systeem geeft voordelen bij het vertalen van de strategieën naar implementatieconcepten, die door de betreffende tool worden gebruikt. Dit komt doordat een specificatie in deze vorm de systeemontwikkelaar bewust maakt van tooleigenschappen, die van belang zijn voor het implementeren van strategieën van redeneren. Door deze vertaling expliciet te maken wordt het inzicht in het systeem vergroot. Dit geeft voordelen bij het onderhoud aan de systemen in de gebruiksfase. Tevens heeft het gebruik van het XYFAR-model het voordeel dat tools onderling kunnen worden vergeleken op geschiktheid voor de implementatie van het betreffende systeem. Vooral door de grotere mogelijkheden om redeneerstrategieën expliciet te formuleren en de noodzaak hiertoe bij het configuratieprobleem, is KES-II een geschiktere tool voor implementatie dan Turbo-Prolog. Wel dient hierbij in het oog te worden gehouden, dat ook andere geschiktheidscriteria een rol kunnen spelen. Bij het criterium "gebruiksvriendelijkheid" bijvoorbeeld zou KES-II ook een betere keuze zijn dan Turbo-Prolog. Bij het criterium "budget" is Turbo-Prolog aan te bevelen.

5 TOEPASSINGSGEBIEDEN VAN KENNISSYSTEMEN

Een systeem met een architectuur zoals in hoofdstuk 2 gedefinieerd, geeft bepaalde voordelen in vergelijking met een systeem, waarbij de verschillende systeemaspecten niet op deze wijze in een architectuur zijn terug te vinden. Om deze voordelen uit te kunnen nutten, moet het toepassingsgebied, waarvoor het kennissysteem wordt ontwikkeld, zekere kenmerken bezitten.

In dit hoofdstuk wordt een beschouwing gegeven van de herkenning van geschikte probleemgebieden. Tevens wordt aangegeven, welke voordelen een kennissysteem met een architectuur, zoals geschetst in het XYFAR-model, biedt ten opzichte van een systeem met een meer "traditionele" architectuur. Hiertoe zal op twee wijzen een kennissysteem worden beschouwd, conform (Davis en Olson, 1984). Hierin wordt aangegeven, dat de volgende ingangen kunnen worden gebruikt bij de beschouwing van informatiesystemen:

- Architectuuringang. Voor de beantwoording van de vraag: wat is het?
- Functionele ingang. Voor de beantwoording van de vraag: wat kun je ermee?
- Combinatie van de beide ingangen. Voor de beantwoording van: waarom kies je een specifieke architectuur als je een specifieke functie aan het informatiesysteem geeft?

De architectuuringang voor de beschouwing van een kennissysteem is al uitgebreid behandeld in hoofdstuk 2 en bijlage A. Dit heeft geresulteerd in het XYFAR-model van een kennissysteem. In paragraaf 5.1. zal een literatuuroverzicht worden gegeven van uitspraken over de herkenning van geschikte probleemgebieden, waarbij wordt uitgegaan van de functionele ingang. Daarna zal in paragraaf 5.2 worden geredeneerd over de voordelen van een kennissysteemarchitectuur en zullen deze voordelen worden vertaald in probleemkarakteristieken. Hierbij wordt uitgegaan van de combinatie van zowel de architectuur- als de functionele ingang. Bij deze beschouwing zal de denktrant uit (Schuwer en Kusters, 1990) en (Kusters en Schuwer, 1991) worden gevolgd.

5.1 GESCHIKTE PROBLEEMGEBIEDEN: FUNCTIONELE INGANG

5.1.1 LITERATUUR

Het doel van deze paragraaf is een overzicht te geven van wat in de literatuur te vinden is over geschikte probleemgebieden.

In (Waterman, 1986) wordt een veelgebruikt "stappenplan" gegeven voor de herkenning van een geschikt probleemgebied. Dit plan bestaat uit drie hoofdstappen, die ieder zijn onderverdeeld in een vijftal deelstappen. Iedere stap bestaat uit een vraag. Na beantwoording van de vragen geeft het stappenplan een conclusie betreffende de geschiktheid van het probleemgebied voor het gebruik van een kennissysteem. In deze context gebruikt Waterman het woord "Expert System". Hieronder verstaat hij een computerprogramma, dat op een relatief klein domein prestaties levert, vergelijkbaar met die van een menselijke expert.

De drie hoofdstappen en de bijbehorende deelstappen zijn de volgende:

- Is het ontwikkelen van kennissystemen *mogelijk*
  - Is voor de oplossing van de taak geen gezond verstand kennis nodig
  - Is voor de oplossing van de taak alleen cognitieve vaardigheid nodig
  - Kunnen de experts hun oplossingsmethoden verwoorden
  - Bestaat er een erkende expert
  - Zijn de experts het eens over de oplossing
  - Is de taak niet te moeilijk
  - Is de taak niet slecht begrepen
- Is het ontwikkelen van kennissystemen *gerechtvaardigd*
  - Geeft oplossing van de taak een hoog rendement
  - Gaat menselijke expertise verloren
  - Is menselijke expertise schaars
  - Is expertise nodig op veel plaatsen
  - Is expertise nodig in een mensvijandige omgeving
- Is het ontwikkelen van kennissystemen *zinvol*
  - Maakt het oplossen van de taak gebruik van symboolmanipulatie
  - Zijn heuristieken nodig voor het oplossen van de taak
  - Is de taak niet te makkelijk
  - Heeft de taak een praktische waarde
  - Is de taak overzienbaar

Veel literatuur die zich bezighoudt met het zoeken naar geschikte probleemgebieden presenteren analoge stappenplannen, die qua detail nauwelijks afwijken van de hier gepresenteerde. Zie bijvoorbeeld (Savory, 1987, pp. 135-149) en (Beckman, 1991). Bij deze beide auteurs wordt overigens niet aangegeven, wat onder een kennis- of expertsysteem moet worden verstaan.

Ook in (Prerau, 1990) wordt een checklist gegeven, waarin de Waterman-lijst is terug te vinden. Hierbij wordt door hem dezelfde definitie van een kennissysteem gehanteerd als Waterman. Prerau houdt echter ook rekening met niet-systeemaspecten, die invloed hebben op de keuze voor een kennissysteemoplossing, zoals managementoverwegingen (bv. van tevoren moet overeenstemming bestaan, dat het probleem ook werkelijk moet worden opgelost) en projectoverwegingen (bv. het project moet niet op het kritieke pad liggen van andere ontwikkelingen).

Chorafas (1987) gebruikt de term "Expert System" in de betekenis van een AI-programma dat het een computer mogelijk maakt een beslissingsproces te ondersteunen, waarbij gewoonlijk expertkennis nodig is. Behalve de hiervoor genoemde hoofdvragen van Waterman onderkent Chorafas de volgende probleemcategorieën, geschikt voor kennissystemen:

- Een klassieke oplossing werkt niet goed.
- Alleen oplosbaar m.b.v. expert-kennis. Een expert wordt hier gedefinieerd als iemand met een grote kennisbank in de vorm van feiten en regels. Verder gebruikt de persoon bij het oplossen van problemen ervaringskennis, die niet of slechts gedeeltelijk in boeken is terug te vinden.
- Problemen, waarbij meerdere acceptabele oplossingen mogelijk zijn (bv. managementproblemen: deze veranderen snel, er worden steeds nieuwe feiten bekend voor een gegeven situatie; references veranderen constant).
- Problemen die constant in ontwikkeling zijn.

Als motivatie waarom voor problemen met deze kenmerken nu juist een kennissysteemoplossing moet worden gekozen, geeft Chorafas het volgende argument: expertsystemen worden over het algemeen als niet gereed beschouwd bij de ingebruikname. Een expertsysteem is constant in ontwikkeling: de kennis wordt voortdurend aangepast.

Veel auteurs gaan uit van probleemkarakteristieken voor het bepalen van de geschiktheid voor een kennissysteemoplossing. In (Jain en Chaturvedi, 1989) wordt een onderzoek naar geschikte probleemdomeinen beschreven. Door de auteurs werd een expertsysteem beschreven als een systeem, dat menselijke kennis gebruikt om problemen op te lossen, waarvoor gewoonlijk menselijke intelligentie is vereist. Het onderzoek had de volgende resultaten. Indien het op te lossen probleem kon worden gedecomponeerd tot kleinere subproblemen of indien bij het probleem duidelijke oorzaak-gevolg relaties konden worden aangegeven, dan was de kans op een succesvol kennissysteem (dat is een kennissysteem, dat ook daadwerkelijk wordt gebruikt) groot. De keuze voor juist een kennissysteem als oplossing werd gemotiveerd met:

- Het probleem was geschikt voor een kennissysteem-applicatie (er werd niet vermeld, hoe dit was gemeten)
- Er was een echte expert beschikbaar
- De wens tot experimenteren
- Het probleem was niet te complex

Ook werden door de respondenten moeilijkheden vermeld, die bij de selectie van het probleem werden ervaren. Een aantal van deze moeilijkheden waren:

- Definitie van de scope van het probleem en de parameters die een rol spelen.
- Vinden van van een expert of groep van experts
- Integratie met andere systemen
- Gebrekkige ontwikkeltools
- Onvoldoende kennis van en ervaring met kennis engineering
- Gebrek aan acceptatie door en medewerking van eindgebruikers

In (Martin en Law, 1988) wordt een kennissysteem beschreven dat gebruikt kan worden om na te gaan of een specifiek probleem geschikt is om aan te pakken met behulp van kennistechnologie. Ze beschrijven daar een kennissysteem niet expliciet. Wel wordt een definitie aangehaald waarin de expliciete representatie van empirische menselijke kennis als kenmerk wordt genoemd. Er blijkt echter nergens uit het artikel welke definitie de auteurs geven aan het begrip kennissysteem. Het beschreven kennissysteem gaat te werk volgens een stappenplan. De onderscheiden stappen zijn de volgende:

- Rechtvaardiging van de applicatie. In het bijzonder rechtvaardiging voor de keuze van juist een kennissysteem in plaats van een traditionele oplossing. De volgende aspecten indiceren een kennissysteem-benadering:
  - Het domein bevat complexe interacties van kennis. Deze interacties kunnen direct worden geïdentificeerd of worden afgeleid en daardoor expliciet gerepresenteerd.
  - Het project bevat zaken als: representatie van onzekerheid, uitleg, archief van expertise, trainingsmiddel, replicatie en verspreiding van kennis.
  - Het domein is voldoende complex om te verwachten dat verschillende iteratieslagen nodig zijn voordat de kennis "op peil" is.

- Bepaal van welk probleemtype sprake is. Hierbij worden de volgende probleemtypen onderscheiden: interpretatie, diagnose, voorspelling, design, planning, monitoring, debugging of reparatie, instructie, control, advies, simulatie.

Deze bepaling is nodig, omdat ieder probleemtype specifieke eisen stelt aan de wijze van kennisrepresentatie en inferentie.

Een soortgelijke probleemtypologie is te vinden in (Breuker et al, 1987). In deze literatuur ontbreekt een definitie van een kennis- of expertsysteem.

In (Kolman et al, 1992) wordt een stappenplan beschreven, waarmee kan worden nagegaan of toepassing van kennistechnologie in een bepaalde situatie zinvol is. Kennistechnologie wordt hier gedefinieerd als het geheel van verrichtingen om kennissystemen tot stand te brengen en de leer ervan. Een kennissysteem wordt in deze publicatie gekenmerkt door:

- Een grote hoeveelheid (gecompliceerde) kennis
- De mogelijkheid tot het geven van verklaringen van oplossingen en aanbevelingen
- De expliciet aangebrachte scheiding tussen de kennisbank en het redeneermechanisme

Bij de methode voor het bepalen van geschiktheid van kennistechnologie in een bepaalde situatie wordt gebruik gemaakt van een classificatie van de situatie naar drie gezichtspunten:

- Omstandigheden: de omstandigheden waarin het systeem wordt ontworpen en/of gebruikt
- Functies: taken die door het systeem moeten worden vervuld
- Kennis: de kenmerken van de informatie cq. kennis die bij het vervullen van de functies van belang zijn

In tabel 5.1 staan per gezichtspunt de kenmerken vermeld, die een rol spelen bij het bepalen van de haalbaarheid van kennistechnologie.

Omstandigheden	Functies	Type kennis
Kennis wordt gebundeld	Afleiden van kennis uit informatie	Kennis is niet volledig
Kennis wordt gedistribueerd	Begeleiden bij gebruik van kennis	Kennis gebaseerd op meervoudige afhankelijkheden
Kennis is moeilijk bereikbaar	Begrenzen van gebruik van kennis	Kennis berust (deels) op ervaring
Gebruiker en bron van kennis van verschillend niveau	Registreren van gebruik van kennis	Kennis bevat veel verbale, niet kwantificeerbare begrippen
Gebruikers hebben onderling een ongelijke hoeveelheid ervaring	Formaliseren van kennis	Kennis bevat onzekerheid
Snelheid is belangrijk	Gebruik van kennis toegankelijk maken	Kennis aan verandering onderhevig
Verloop onder gebruikers	Het doen toenemen van kennis	
Kennis dreigt verloren te gaan	Vergroten van betrouwbaarheid van het gebruik van kennis	

Tabel 5.1 Overzicht van kenmerken per gezichtspunt

Per kenmerk is een criterium opgesteld waarmee kan worden nagegaan of het betreffende kenmerk van belang is voor de te beoordelen situatie. Afhankelijk van dit belang wordt een waardering 0 (geen belang), 1 (secundair belang) of 2 (primair belang) aan een kenmerk gegeven. (Van primair belang is sprake, indien het kenmerk expliciet vermeld is in de probleemspecificatie. Van secundair belang is sprake, indien het kenmerk wel relevant is, maar niet expliciet vermeld staat in de probleemspecificatie.) Het oordeel "wel of geen kennistechnologie" kan nu worden gegeven na optelling van alle waarderingen. Indien de som van alle waarderingen tussen 12 en 20 ligt, dan is toepassing van kennistechnologie zinvol. De auteurs merken overigens op, dat deze grenzen enigszins arbitrair zijn.

Buchanan en Shortliffe (1984) gaan in op toepasbaarheid van op regels gebaseerde kennissystemen. (Ter herinnering: de definitie van een kennissysteem door Buchanan en Shortliffe is gegeven in hoofdstuk 2.2). Ze onderkennen drie domeinkarakteristieken, die indiceren of regelgebaseerde systemen geschikt zijn:

1. Het probleem bestaat uit een verzameling onafhankelijke toestanden. Voorbeeld: enkele medisch domeinen, waarbij er veel mogelijke probleemtoestanden zijn en relatief weinig mogelijke te nemen acties.
2. De control flow bestaat uit een verzameling onafhankelijke acties.
3. De kennis betreffende het probleem kan worden geformuleerd onafhankelijk van de wijze, waarop ze wordt gebruikt.

5.1.2 EVALUATIE

Bij het overzicht uit de literatuur in paragraaf 5.1.1 kan het volgende worden geconstateerd:

Veel probleemtypologieën gaan uit van de intelligentie-definities van een expertsysteem, zoals in hoofdstuk 2 werd beschreven. Er wordt geconstateerd, dat veel experts zich met problemen bezighouden van het type zoals hierboven beschreven in (Martin en Law, 1988). Deze probleemtypen zijn dan "dus" geschikt om opgelost te worden met kennissystemen. De diverse probleemtypologieën zijn onderling strijdig. Zo wordt bij het onderzoek van Jain en Chaturvedi als ervaring geconstateerd, dat een probleem niet te complex mag zijn. Martin en Law stellen echter, dat het domein complexe interacties van kennis moet bevatten.

De hiervoor gegeven constatering dat veel probleemtypologieën uitgaan van intelligentie-definities is een verklaring voor het feit, dat bij deze probleemtypologieën geen verklaring wordt gegeven *waardoor* een systeemarchitectuur, zoals die uit het XYFAR-model, een geschikte architectuur is voor het oplossen van een expertprobleem. Men gaat immers niet uit van een dergelijke architectuur in hun beeld van een kennissysteem. Zo'n verklaring wordt wel gegeven in (Altenkrüger, 1990). Daar wordt een koppeling gemaakt tussen wijzen van redeneren, die in verschillende implementaties van kennistechnologie is te vinden en de wijze van oplossen voor verschillende probleemtypen. Zo constateert hij, dat voorwaarts redeneren vooral van belang is bij onder meer simulatie, planning en ontwerp. Achterwaarts redeneren komt vooral voor bij classificatie, analyse en diagnose.

Bij de twee stappenplannen van Waterman en Kolman et al valt het op, dat deze weinig overeenkomsten vertonen. Tabel 5.2 geeft aan, welke overeenkomsten en verschillen er zijn te constateren.

Waterman → Kolman et al ↓	Kennissysteem mogelijk	Kennissysteem gerechtvaardigd	Kennissysteem zinvol
Omstandigheden (O)	Andere O-factoren bij Waterman	Deels andere O-factoren bij Waterman	Andere O-factoren bij Waterman
Functies (F)	Niet aanwezig bij Waterman	Niet aanwezig bij Waterman	Niet aanwezig bij Waterman
Kennis (K)	Andere K-factoren bij Waterman	Niet aanwezig bij Waterman	Twee K-factoren overeenstemmend

Tabel 5.2 Vergelijking tussen stappenplannen

De factoren die een geschikt toepassingsgebied indiceren volgens Waterman zijn geen van alle te classificeren als Functie-factoren, zoals Kolman et al die onderkennen. Daarnaast bestaat weinig overeenkomst tussen de O- en K-factoren uit beide stappenplannen. De beide stappenplannen lijken daarom op het eerste gezicht aanvullend aan elkaar.

Er zijn enkele bezwaren aan te voeren tegen de beschreven visies welke problemen wel of niet geschikt zijn voor kennissystemen:

1. De stappenplannen van Waterman en Kolman et al hebben als nadeel, dat enkele stappen pas achteraf kunnen worden beoordeeld (bijvoorbeeld of experts in staat zijn hun kennis te verwoorden of dat gebruik wordt gemaakt van heuristieken). Het maken van een eerste prototype als een vorm van een haalbaarheidsstudie kan mogelijk een oplossing bieden voor dit probleem. Een prototype geeft echter geen compleet beeld van het hele probleemgebied en dus is dit maar een beperkt antwoord.
2. Omdat niet wordt uitgegaan van een systeemarchitectuur wordt nergens een indicatie gegeven of *alle* problemen uit bovenstaande typologieën inderdaad geschikt zijn om met behulp van een kennissysteem te worden aangepakt. Deels geven de stappenplannen van Waterman en Kolman et al uitsluitsel. Zo kunnen eenvoudige diagnoseproblemen (weinig regels, duidelijke struktuur) beter met traditionele

> technieken worden opgelost. Het blijft echter onbeantwoord *of* een systeem met de architectuur uit het XYFAR-model inderdaad geschikter is om dit type problemen op te lossen dan een traditionele architectuur. Tevens wordt niet aangegeven, *waarom* zo'n architectuur beter is.

Auteurs, die niet uitgaan van een architectuurdefinitie van een kennissysteem, zijn veelal te classificeren als aanhanger van de, in hoofdstuk 2 genoemde, processchool. In feite houden ze zich bij het zoeken naar geschikte probleemtypen bezig met de vraag welke probleemtypen geschikt zijn, om de kennis ervan nader te expliciteren. In deze studie zal het begrip *kennistechnologie* worden gebruikt om dit te beschrijven. In feite spelen bij het zoeken naar geschikte probleemtypen twee vragen:

1. Welke probleemtypen lenen er zich voor om met behulp van *kennistechnologie* te worden aangepakt en welke toegevoegde waarde levert dit?
2. Welke probleemtypen lenen er zich voor om met behulp van een *kennissysteem* te kunnen worden opgelost en welke toegevoegde waarde levert dit?

Om de eerste vraag te kunnen beantwoorden, zal eerst nader worden omschreven, wat in dit kader onder kennistechnologie wordt verstaan.

In (Mars, 1991) wordt (in een voetnoot) de volgende omschrijving van kennistechnologie gegeven (p. 5):

> "We zouden kunnen onderscheiden: kennistechniek als het geheel van de verrichtingen om kennissystemen tot stand te brengen, en kennistechnologie als de leer van de kennistechniek."

Volgens deze definitie zou een kennistechnologische benadering van systeemontwikkeling het doel hebben om de struktuur van een kennissysteem (in de "architectuur-zin") te ontwikkelen. Het resultaat hoeft echter niet per se een geïmplementeerd kennissysteem te zijn.

In (Heijne den Bak en Curwiel, 1989) wordt kennistechnologie (KT) in relatie tot informatietechnologie (IT) omschreven als (p. 101):

> "IT is a set of related methods and techniques which are especially suited for analyzing, modelling, implementing and operating data-handling processes for which a manageable algorithm (an explicitly formulated, step-by-step decision procedure) can be found.
>
> KT is a set of related methods and techniques especially suited for analyzing, modelling, implementing and operating data-handling processes for which no manageable algorithm can be found by means of IT."

In deze definitie wordt de betekenis van kennistechnologie gelegd bij het hanteerbaar maken van niet geheel "begrepen" gegevensverwerkende taken. Wanneer deze taken mbv. informatietechnologie worden ondersteund, worden die "onbegrepen" taken niet door het informatiesysteem uitgevoerd, maar door de gebruiker van het systeem. Bij een kennistechnologie-visie op systeemontwikkeling wordt geprobeerd een deel van die "onbegrepen" taken door het systeem te laten uitvoeren. Ook hier betekent een kennistechnologie-visie op systeemontwikkeling, dat kennis van taken, die bij een IT-visie impliciet beschreven blijft, wordt geëxpliciteerd. Het resultaat van zo'n systeemontwikkeling hoeft, evenals bij Mars, echter niet per se te leiden tot een kennissysteem in de zin van het XYFAR-model.

De laatste omschrijving van kennistechnologie geeft weer, dat het begrip kennistechnologie niet statisch is. Door het expliciteren van eerder impliciet gelaten taken behoren alle overeenkomstige latere systemen tot de informatietechnologie. Deze constatering is ook te trekken uit de definitie van AI, zoals die te vinden is in (Mars, 1991) (p. 7):

> "Artificial Intelligence is het deelgebied van de informatica waarin computerprogramma's worden vervaardigd voor het uitvoeren van taken die - naar tien jaar geleden algemeen werd aangenomen - intelligentie vereisen."

Uit de twee definities voor kennistechnologie zijn de volgende aspecten (direct of meer indirect) te halen, die het begrip kennistechnologie beschrijven:

- Kennistechnologie besteedt veel aandacht aan het expliciet maken van kennis. Het resultaat van dit expliciet maken van kennis bestaat minimaal uit een beschrijving van de kennis, die een rol speelt in het betreffende domein.
- Kennistechnologie bevat een aantal technieken voor het achterhalen en weergeven van kennis. Als voorbeeld kan het vierlagenmodel uit de methode KADS (zie hoofdstuk 3) worden genoemd. Ook diverse technieken voor het uitvoeren van een kenniselicitatie behoren tot kennistechnologie.

- Kennistechnologie bevat een aantal produkten, die geschikt zijn voor het expliciet weergeven van kennis, zoals een kennissysteemshell.
- Kennistechnologie kan leiden tot een kennissysteem in architectuur-zin, maar dit is niet per se nodig. Andersom vraagt het bouwen van een kennissysteem altijd om een kennistechnologische visie van ontwikkelen, omdat dit type systeem expliciet geformuleerde kennis bevat.

Door deze aspecten wordt een beeld van kennistechnologie gegeven, zonder het eenduidig te beschrijven. Binnen het kader van deze studie is zo'n eenduidige beschrijving echter niet noodzakelijk. Het blijft ook een open vraag of, door onder andere het niet-statische karakter van kennistechnologie, een eenduidige definitie wel mogelijk is. De toegevoegde waarde van een kennistechnologische aanpak van systeemontwikkeling voor deze klasse van problemen ligt in het verkregen model van de kennis, waardoor de probleemoplostechnieken gestruktureerd worden en daardoor het betreffende probleem inzichtelijker is geworden.

De in 5.1.1 door verschillende auteurs gegeven probleemclassificaties geven weer, welke probleemtypen er zich voor lenen om met behulp van *kennistechnologie* aan te pakken. Veelal ontbreken voor problemen van de genoemde typen vaste oplossingsalgoritmen. Vaak is de kennis om een probleem van zo'n type op te lossen van tevoren niet geformaliseerd. Dit is in het bijzonder het geval in de situatie waarbij menselijke ervaringskennis een belangrijke rol speelt.

Indien auteurs bij een geschiktheidsbeoordeling uitgaan van een intelligentie-definitie van een kennissysteem, dan wordt door die auteurs vaak geen onderscheid gemaakt tussen kennis*technologie* en kennis*systeem*. De betreffende auteurs geven dan echter geen selectiecriteria weer voor het toepassen van een kennissysteemoplossing (dus een systeem met een architectuur volgens het XYFAR-model), maar voor een kennistechnologische visie op systeemontwikkeling. Bij de auteurs, die in 5.1.1 werden besproken, geven alleen (Kolman et al, 1992) expliciet aan, dat geschiktheid voor kennistechnologie wordt onderzocht. Ze laten echter in het midden of dit ook geschiktheid voor een kennissysteem inhoudt.

In de volgende paragraaf zal worden aangegeven, welke probleemkarakteristieken een kennissysteemoplossing (in architectuur-zin) rechtvaardigen. Tevens zal worden aangegeven, waaruit de toegevoegde waarde bestaat van zo'n kennissysteemoplossing ten opzichte van een traditionele (IT-)oplossing.

5.2 GESCHIKTE PROBLEEMGEBIEDEN: GECOMBINEERDE INGANG

Een keuze voor een bepaald type informatiesysteem zou gerechtvaardigd moeten worden door de toegevoegde waarde, die dat type systeem heeft ten opzichte van andere typen informatiesystemen. Deze toegevoegde waarde kan worden ontleend aan de wijze, waarop het systeem voldoet aan de eisen, die eraan gesteld worden. In de terminologie van Bemelmans (1991) wordt bij het evalueren van de toegevoegde waarde van een informatiesysteem onderscheid gemaakt in:

- het voldoen aan functionele eisen. Hier gaat het om al die specificaties die aangeven welke gegevens verwerkt en verstrekt moeten worden (wat moet het systeem doen).
- het voldoen aan niet-functionele eisen, vaak prestatie-eisen of kwaliteitseisen genoemd. Deze betreffen de voorwaarden waaronder gegevensverwerking en -verstrekking zich dienen te voltrekken (hoe goed doet een systeem iets).

Om de toegevoegde waarde van een kennissysteem te beoordelen zal worden uitgegaan van de specifieke architectuur, zoals die is vastgelegd in het XYFAR-model. Deze specifieke architectuur zou kunnen bijdragen in het beter kunnen voldoen aan zowel de functionele als de niet-functionele eisen.

De specifieke architectuur zal *in theorie* niets toevoegen aan mogelijke functionaliteit, die bereikt kan worden. Alle functies, die door een kennissysteem kunnen worden uitgevoerd, zijn ook mogelijk met behulp van informatiesystemen met een traditionele architectuur. In de praktijk kan het voorkomen dat het realiseren van een bepaalde functionaliteit wel degelijk afhangt van het type software, dat wordt gebruikt. Zo zal het realiseren van een DBMS, geheel geschreven in assembler, waarschijnlijk niet mogelijk zijn. Dit hangt samen met de complexiteit, die door de mens nog kan worden bevat bij de realisatie van een oplossing. In (Lehmann en Belady, 1985) wordt complexiteit van een probleem omschreven als:

> De complexiteit van een probleem is de mate van onzekerheid, die in het oplossen van het probleem besloten ligt. De mate van onzekerheid van het oplosproces van een probleem is evenredig met de variëteit aan objecten, die een rol spelen bij het probleem en zijn oplossing, het aantal toestanden waarin een (oplos)proces zich kan bevinden, het gebrek aan a priori informatie over de relatieve relevantie van objecten voor de volgende op te lossen taak en de variëteit aan informatie, die nodig is op elk tijdstip van het oplosproces.

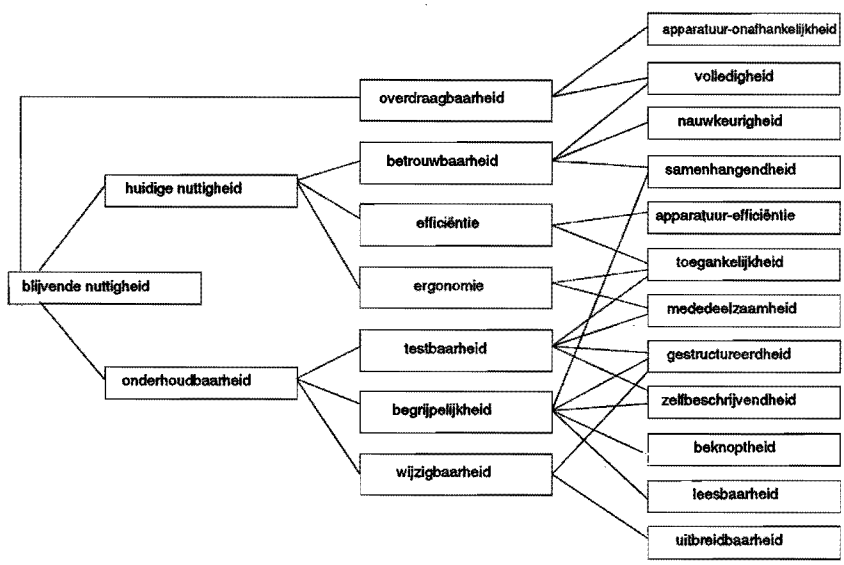
Toegepast op de realisatie van een systeem kan een specifiek type software handvatten bieden bij het reduceren van de complexiteit van een probleemoplosproces door voor de gebruiker van de software een of meer van de genoemde complexiteitsbepalende factoren te verkleinen. De specifieke systeemarchitectuur van een kennissysteem vermindert de complexiteit door het aantal toestanden, waarin het oplosproces zich kan bevinden en die door de systeemontwerper moet worden geëxpliciteerd, te verkleinen. Tevens wordt door het systeem, via de metaregels in de inferentieprocedure, de relatieve relevantie van objecten voor de volgende op te lossen taak, automatisch bepaald. Ook deze oplostaak hoeft dus door de gebruiker niet tot het laatste detail te worden gespecificeerd.

In hoofdstuk 2 werd al aangegeven, dat voor sommige auteurs het kunnen geven van uitleg over de door het systeem gemaakte redenering een kenmerk is van een kennissysteem. Ook (Kolman et al, 1992) geven deze functionaliteit expliciet als kenmerk van een kennissysteem. In hoofdstuk 2.4 staat reeds aangegeven, dat de redeneerketen, die door het systeem wordt gegenereerd, als basis kan dienen voor het geven van uitleg. In (Wognum, 1990) is een vertaling aangegeven van zo'n mechanische redeneerketen (gebaseerd op het resolutie-principe) naar een redeneerketen, die voor mensen bevattelijker is (gebaseerd op een vorm van redeneren, die "natural deduction" wordt genoemd). De functionaliteit "het geven van uitleg" is *praktisch* gesproken vanwege de complexiteit alleen haalbaar door gebruik te maken van een systeemarchitectuur, zoals beschreven staat in het XYFAR-model.

Vaak is het mogelijk, dat een bepaalde functionaliteit *makkelijker* kan worden bereikt door de architectuur. Zo zal het afleiden van nieuwe gegevens uit reeds bestaande gegevens met behulp van afleidingsregels door een kennissysteem "automatisch" beter worden uitgevoerd. De architectuur van een kennissysteem is geschikt voor juist die functionaliteit (zie het XYFAR-model). Andersom zal het meer moeite kosten om met behulp van een kennissysteem complexe berekeningen uit te voeren, waarbij een verandering van een gegeven een kettingreactie van veranderingen voor andere gegevens tot gevolg heeft. Dit type problemen leent zich er beter voor om opgelost te worden met behulp van andere hulpmiddelen zoals een spreadsheet.

De hierboven gebruikte terminologie ("makkelijker", "beter") indiceert dat toegevoegde waarde met name gezocht moet worden in het beter kunnen voldoen aan niet-functionele eisen. Om na te gaan, aan welke niet-functionele eisen een kennissysteem beter kan voldoen zal allereerst een classificatie van deze eisen worden gepresenteerd. Hiertoe wordt de traditionele kwaliteitsboom gepresenteerd, waarin te stellen eisen in onderling verband zijn weergegeven. Er is hier gekozen voor de boom uit (Boehm et al, 1978). (De hier gebruikte vertaling van de begrippen uit deze boom is te vinden in (Paulussen et al, 1992)). In deze

boom worden eisen onderscheiden op een viertal abstractieniveaus. Het hoogste abstractieniveau bevat de eis "blijvende nuttigheid". Deze eis kan worden gedecomponeerd in twee deeleisen op een lager astractieniveau: "huidige nuttigheid" en "onderhoudbaarheid". Ieder van de deeleisen kunnen uiteindelijk, via een tussenniveau, worden opgesplitst in een aantal primitieven. Figuur 5.1 geeft een overzicht van de kwaliteitsboom. Hierin moet een verbinding tussen twee eisen van links naar rechts worden gelezen als "wordt bepaald door".



Figuur 5.1 De kwaliteitsboom van Boehm

(In de figuur is ook te zien, dat het begrip "boom" in feite verkeerd is: de eisen op het laagste niveau kunnen meerdere "ouders" hebben. Dit is bij een boom niet mogelijk.) De eisen op het laagste niveau zijn als volgt omschreven:

Apparatuur-onafhankelijkheid	De mate, waarin een softwareprodukt kan worden geëxecuteerd op andere hardware dan die, waarvoor het is ontwikkeld.
Volledigheid	De mate waarin alle delen van een softwareprodukt aanwezig zijn en ieder van de delen volledig is ontwikkeld. Dit impliceert onder meer "self-containedness" en robuustheid.

Nauwkeurigheid	De mate, waarin de output van een softwareprodukt voldoet aan de bedoeling ervan in de specificaties.
Samenhangendheid	De mate, waarin een softwareprodukt uniformiteit in notatie, terminologie en symbolen bezit (interne samenhangendheid). De mate, waarin de inhoud van een softwareprodukt kan worden vergeleken met de requirements (externe samenhangendheid).
Apparatuur-efficiëntie	De mate, waarin een softwareprodukt zijn taken uitvoert met zo zuinig mogelijk gebruik van resources.
Toegankelijkheid	De mate, waarin het mogelijk is selectief componenten van een softwareprodukt te gebruiken.
Mededeelzaamheid	De mate, waarin het eenvoudig mogelijk is om van een softwareprodukt de input te specificeren en de mate, waarin het mogelijk is om output te verschaffen, waarvan de vorm en inhoud eenvoudige verwerking mogelijk maakt en nuttig is.
Gestructureerdheid	De mate, waarin een softwareprodukt een precieze, duidelijk omschreven struktuur bezit, samengesteld uit de afzonderlijke componenten.
Zelfbeschrijvendheid	De mate, waarin een softwareprodukt voldoende informatie voor een lezer bevat om de doelen, aannames, beperkingen, input, output, componenten en status te achterhalen.
Beknoptheid	De mate, waarin geen overdadige informatie in een softwareprodukt aanwezig is.
Leesbaarheid	De mate, waarin het mogelijk is om door het lezen van de code van een softwareprodukt de functie ervan te achterhalen.
Uitbreidbaarheid	De mate, waarin een softwareprodukt eenvoudig uitbreidbaar is.

De specifieke architectuur van een kennissysteem maakt het mogelijk om beter aan de volgende eisen op het laagste niveau van de boom te voldoen:

Samenhangendheid	De expliciete definitie van de gebruikte kennis geeft mogelijkheden tot een betere vergelijking van de implementatie van deze kennis met de specificatie van de kennis (externe samenhangendheid).
Toegankelijkheid	Doordat de verschillende systeemaspecten expliciet zijn gedefinieerd en onderscheiden is het mogelijk ieder van de resulterende componenten afzonderlijk te benaderen.
Gestructureerdheid	De opdeling van het systeem in de diverse, goed-gedefinieerde componenten geeft een duidelijke structuur aan het systeem.

Zelfbeschrijvendheid	De expliciete definitie van ieder van de componenten enerzijds en de vorm, waarin de definitie wordt gegeven (in het bijzonder de declaratieve vorm van de regels in de regelbank) maakt het systeem in hoge mate zelfbeschrijvend. In hoofdstuk 4 is echter ook vermeld, dat additionele informatie nodig blijft.
Leesbaarheid	Het niet vermengen van kennis en afleidingsregels en de precieze definitie van ieder van de componenten maakt het mogelijk uit de systeemcode informatie betreffende doelen, input etc. te achterhalen.
Uitbreidbaarheid	Door de modulaire opbouw van het systeem en de scheiding van systeemaspecten in de diverse modules wordt de uitbreidbaarheid groter dan wanneer de systeemaspecten vermengd zijn (minder side effects).

Uit de invloed die een kennissysteemarchitectuur heeft op de eisen op het laagste niveau van de boom, kan een abstractieniveau hoger worden gezien, dat vooral *testbaarheid*, *begrijpelijkheid* en *wijzigbaarheid* van een systeem beter te realiseren zijn met een kennissysteemarchitectuur. Weer een niveau hoger blijkt, dat de uiteindelijke toegevoegde waarde van een kennissysteem ligt in het beter kunnen voldoen aan de eis van *onderhoudbaarheid* van een systeem. Anders geformuleerd geeft het gebruik van een kennissysteem als een oplossing voor een zeker probleem voordelen op het gebied van de onderhoudbaarheid van de kennis in het systeem. In (Davis en Buchanan, 1977) wordt dit eveneens geconstateerd ten aanzien van metaregels. Zij zien als voordeel van het expliciet aanwezig zijn van metaregels een conceptueel "zuiverder" organisatie van kennis en (daardoor) een eenvoudiger onderhoudbaarheid van redeneerstrategieën.

Opgemerkt dient te worden, dat onderhoudbaarheid slechts één van de niet-functionele eisen is die aan een te ontwikkelen systeem kan worden opgelegd. Bij de afwegingen, die uiteindelijk leiden tot het al dan niet gebruiken van een kennissysteem moeten ook de andere eisen worden betrokken. In het bijzonder kan een kennissysteem moeilijk voldoen aan de eis van apparatuur-efficiëntie ((Liebowitz, 1991), (Harmon en Maus, 1988), (Wognum en Lippolt, 1991)). Indien deze eis zwaarder weegt dan onderhoudbaarheid van kennis, dan is een kennissysteem als wijze van oplossen niet aan te bevelen.

Dat het niet-functionele eisen zijn, die uiteindelijk het al dan niet kiezen van een kennissysteem als een oplossing bepalen, is niet nieuw. Zo wordt bij traditionele systeemontwikkeling de keuze van een 4GL in plaats van een 3GL uiteindelijk gemotiveerd met argumenten als "betere aanpasbaarheid" of "snellere ontwikkeling mogelijk".

Probleemtypen, die geschikt zijn om met behulp van een kennissysteem op te lossen, moeten karakteristieken bezitten, die aanduiden, dat onderhoudbaarheid van de kennis een belangrijke eis is. Volgens het XYFAR-model is deze kennis vervat in een database (de gegevens), een regelbank en een redeneerprocedure.

Onderhoudbaarheid van gegevens betekent dat gebruik moet worden gemaakt van een DBMS. Het gebruik van een DBMS wordt gerechtvaardigd indien (Everest, 1986):

- de gegevens vaak wijzigen
- de omvang van de database groot is
- gegevens door meer gebruikers en applicaties tegelijk worden gebruikt

Analoge overwegingen kunnen ook worden gemaakt voor de regelbank en de redeneerprocedure.

Onderhoudbaarheid van een verzameling regels kan een probleem zijn, indien vaak *veranderingen* in die regels optreden, i.e. wanneer de kennis niet robuust is. Dit kan bijvoorbeeld het geval zijn, indien een kennisgebied nog in ontwikkeling is. Ook kan de eerste gebruiksfase van een systeem deze karakteristiek bezitten. Door gebruik van het systeem komen dan ontbrekende regels of tekortkomingen in de redeneerprocedure boven tafel. Dit kan op termijn leiden tot een situatie, waarbij deze wijzigingen niet of nauwelijks meer optreden, ofwel waarbij de verzameling kennis robuust is geworden. Indien andere niet-functionele eisen dan hogere prioriteit krijgen, kan dat zelfs leiden tot een nieuwe realisatie, waarbij de kennissysteemarchitectuur wordt verlaten, omdat die geen toegevoegde waarde meer bezit en het met deze architectuur moeilijk blijkt te zijn aan de andere relevante niet-functionele eisen te voldoen.

Het aantal wijzigingen in de verzameling kennis is op zichzelf geen voldoende motivatie voor de keuze van een kennissysteemoplossing. Indien de verzameling regels *groot* en/of *complex* is of indien de redeneerprocedure ingewikkeld is, kan dat aanleiding zijn een kennissysteemoplossing te overwegen. In deze situatie is het moeilijk om het effect van een verandering aan een regel op de andere regels te overzien, zeker indien aan de regelbank geen structuur wordt gegeven (zoals in het MYCIN-voorbeeld door de contextboom). Indien de regels ook nog zijn vervat in programmacode, waar tevens sprake is van redeneeractiviteiten, dan wordt het bijzonder moeilijk het effect van een wijziging te overzien, als al kan worden bepaald, waar de wijziging moet plaatsvinden. Indien de grootte of complexiteit van de regelbank of redeneerprocedure toeneemt, ontstaat een grotere behoefte aan overzicht over deze componenten. Het is eenvoudiger dit overzicht te verkrijgen

indien de kennis geëxpliciteerd wordt en gescheiden wordt gerepresenteerd in een regelbank en een redeneerprocedure, dan wanneer het is "verstopt" in programmacode.

Ook de wijze, waarop gebruik wordt gemaakt van de regelbank kan invloed hebben op de uiteindelijke keuze van een kennissysteem. Indien de regels in de regelbank door meer gebruikers of applicaties tegelijk worden gebruikt, ontstaat een behoefte aan soortgelijke functionaliteit t.a.v. beveiliging en toegangsmogelijkheden voor de regelbank als bij databases door een DBMS wordt geboden. Er kan bijvoorbeeld de situatie ontstaan, dat door gebruik van een of meerdere regels in de regelbank gegevens worden afgeleid, die voor de betreffende gebruiker of applicatie moeten worden afgeschermd. Hoewel deze situatie nog niet veel voorkomt (de meeste geïmplementeerde kennissystemen zijn stand-alone systemen) kan het in de toekomst een belangrijke overweging zijn voor de keuze van een oplossing, waarbij *kennisbeheer* noodzakelijk wordt.

Behalve de aandacht voor beveiliging en toegangsmogelijkheden vergt de ontwikkeling van applicaties met een gelijktijdig gebruikte regelbank ook meer specifieke aandacht voor een aantal potentiële problemen en valkuilen. In hoofdstuk 6 zal hier nader op worden ingegaan.

De hiervoor gegeven probleemkarakteristieken die onderhoudbaarheid van kennis indiceren, zijn deels terug te vinden bij de auteurs, die in hoofdstuk 5.1.1 werden genoemd. Tabel 5.3 geeft hiervan een overzicht.

Auteur	Probleemkarakteristiek		
	Veel verandering	Veel/complexe kennis	Gelijktijdig gebruik
Waterman	-	Taak niet te makkelijk	-
Chorafas	Probleem constant in ontwikkeling	-	-
Jain & Chaturvedi	-	NIET complex	-
Martin & Law	-	Complexe interacties van kennis	-
Kolman et al	Kennis aan verandering onderhevig & kennis groeit in de loop der tijd	Kennis gebaseerd op meervoudige afhankelijkheden & kennis bevat veel verbale begrippen	-
Buchanan & Shortliffe	-	-	-

Tabel 5.3 Vergelijking probleemkarakteristieken

Geen van de auteurs noemt het gelijktijdig gebruik van kennis expliciet. Door Kolman et al wordt ook de eerder onderkende functionele eis van het geven van uitleg genoemd ("Gebruik van kennis toegankelijk maken"). Zoals eerder al werd opgemerkt is de toegevoegde waarde van een kennissysteem in de architectuur-betekenis voor de overige door de auteurs genoemde probleemkarakteristieken niet aan te geven.

De probleemkarakteristieken, die de keuze voor het gebruik van een DBMS bepalen en de karakteristieken, die de keuze voor een kennissysteemoplossing indiceren, kunnen worden gecombineerd. Er worden dan vier situaties onderscheiden, waarbij iedere situatie gekarakteriseerd wordt door het al dan niet aanwezig zijn van een extra onderhoudsinspanning voor de gegevens enerzijds en de regelbank en redeneerprocedure anderzijds.

- Situatie 1
  Zowel voor de gegevens als voor de regels en de redeneerprocedure ontbreekt de noodzaak voor een grote onderhoudsinspanning. In deze situatie hebben kennissystemen geen bijzondere voorkeur als implementatie-oplossing, bezien vanuit het aspect onderhoudbaarheid.
- Situatie 2
  Er bestaat behoefte aan gegevensbeheer, maar geen behoefte aan een extra onderhoudsinspanning voor regels en redeneerprocedure. In deze situatie komt een DBMS in aanmerking als oplossing.
- Situatie 3
  Er bestaat behoefte aan een grote onderhoudsinspanning voor de regels en/of de redeneerprocedure, maar geen behoefte aan gegevensbeheer. In deze situatie komen kennissystemen in aanmerking als oplossing.
- Situatie 4
  Er bestaat behoefte aan een grote onderhoudsinspanning voor zowel de gegevens als de regelbank en/of redeneerprocedure. In deze situatie bieden zowel de huidige generatie DBMS'en (onvoldoende mogelijkheden voor de regelbank en/of redeneerprocedure) als de huidige generatie kennissystemen (onvoldoende mogelijkheden voor gegevensbeheer) geen oplossing. In feite is er dan behoefte aan een Knowledge Base Management Systeem (KBMS).

Een *KBMS* wordt hier gedefineerd als een geheel aan computerprogramma's, waarmee

- de componenten van een kennissysteem kunnen worden gedefinieerd
- de database kan worden onderhouden
- de regelbank kan worden onderhouden
- de redeneerprocedure kan worden onderhouden
- de beveiliging van de componenten van een kennissysteem kan worden gerealiseerd.

Merk op, dat de functionaliteit van een KBMS logischerwijze die van een DBMS omvat. Een voorbeeld van een implementatie van een KBMS is het systeem, zoals beschreven in (Herwijnen et al, 1990).

Tabel 5.4 geeft een overzicht van de vier situaties. Hierbij betekent "sterk" en "zwak" resp. de aanwezigheid dan wel afwezigheid van de onderhoudsinspanning voor database of regelbank en/of redeneerprocedure.

		database	
		zwak	sterk
regelbank/ redeneerprocedure	zwak	niet DBMS/KBMS (situatie 1)	DBMS (situatie 2)
	sterk	Kennissysteem (situatie 3)	KBMS (situatie 4)

Tabel 5.4 Keuze voor DBMS, kennissysteem of KBMS

Samenvattend kunnen situaties worden onderscheiden, waarbij informatietechnologie (IT), kennistechnologie (KT), een kennissysteem (KS) of een KBMS kunnen worden gebruikt. De samenhang tussen deze situaties wordt grafisch weergegeven in figuur 5.2.

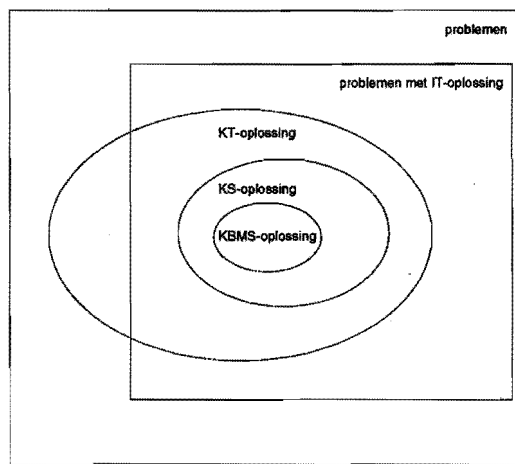


Fig. 5.2 Samenhang tussen verschillende typen oplossingen

In het voorgaande werd specifiek onderhoudbaarheid van de kennis als toegevoegde waarde genoemd van een kennissysteemarchitectuur. In de praktijk worden echter ook andere voordelen genoemd. In (Liebowitz, 1991) zijn ongeveer 160 beschrijvingen van implementaties van kennissystemen te vinden. Bij ongeveer 20 van die beschrijvingen is aangegeven, waarom het betreffende systeem een succesvolle implementatie kende (ofwel:

waarom het systeem ook werkelijk wordt gebruikt) en welke toegevoegde waarde het systeem bezit. Indien aangegeven hadden de applicaties een grote regelbank (500-7000 regels) en ondersteunden ze een complexe taak (meestal scheduling). De complexiteit wordt dan bepaald door het grote aantal regels en relaties tussen de gegevens, met de mogelijkheid van een combinatorische explosie bij het berekenen van een uitkomst. De aard van die applicaties is dus zodanig, dat kennisbeheer een sterke eis is. Deze beschrijvingen kunnen daarom mede worden beschouwd als een validatie uit de praktijk voor de hiervoor getrokken conclusies over toepassingsgebieden en toegevoegde waarde van een kennissysteem. Uit de beschrijvingen kunnen de volgende conclusies worden getrokken.

- Als succesfactor wordt diverse keren de grotere aanpasbaarheid van de regelbank genoemd. Wel wordt opgemerkt, dat bij grotere regelbanken (500 regels of meer) een struktuur over die regelbank nodig is. Zo'n struktuur kan bijvoorbeeld bestaan uit een hiërarchische ordening van de regels. Zonder zo'n struktuur gaat het overzicht over de regelbank verloren. Men is dan niet in staat te beoordelen of een nieuw toegevoegde regel in conflict is met bestaande regels of mogelijk overbodig is. Het aanbrengen van een struktuur lijkt echter alleen mogelijk indien de op te lossen taak decomponeerbaar is. De decompositie geeft dan een "natuurlijke" opdeling van de regelbank door aan ieder van de deeltaken een deel van de regelbank te koppelen.
- Meermalen was alleen in een eerste gebruiksfase sprake van een behoefte aan aanpasbaarheid van de regelbank, omdat de kennis nog niet uitontwikkeld was. Na verloop van tijd stabiliseerde de kennis zich. In een geval werden eisen gesteld aan de responsetijd en was men genoodzaakt het systeem te herschrijven in een conventionele 3GL (de taal C). Hierbij werd de aanpasbaarheid veel kleiner, maar deze eis was niet meer zo belangrijk vanwege het uitontwikkeld zijn van de kennis.
- Meermalen werd als grote voordeel van een shell genoemd, dat deze een groot programmeergemak biedt. In feite wordt het met die software mogelijk *declaratief* te programmeren, zonder zorgen te hebben over de procedurele afhandeling. De toolanalyse uit hoofdstuk 4 heeft echter aangetoond, dat dit "ideaalbeeld" niet voor de volle 100% haalbaar is, doordat vaak redeneeraspecten ofwel impliciet blijven (volgorde van regels is belangrijk) ofwel opgenomen moeten worden in de regelbank, met als gevolg een mindere transparantie van de regelbank.
- Het gebruik van algemene shells voor specifieke problemen stuit soms op moeilijkheden doordat het redeneermechanisme niet of heel moeilijk kan worden beïnvloed ofwel door een starre wijze van kennisrepresentatie. Dit kan zich uiten in een kloof tussen de taalkundige expressies die in het domein gewenst zijn en die, die door de shell worden geboden. Hierdoor wordt een correctheidstest lastig, wat weer gevolgen heeft voor het onderhoud van het systeem.

Er worden ook andere slaag- en faalfactoren genoemd, die niet terug te voeren zijn op de specifieke kennissysteemarchitectuur. Deze factoren zijn:

Slaagfactoren

- Herbruikbaarheid van reeds aanwezige data in databases.
- Mens-machine interface, die niet afwijkt van die uit traditionele systemen.
- Overdracht mogelijk van het resultaat van het kennissysteem naar een traditioneel systeem.
- Applicatie moet kritisch zijn voor de organisatie. Dat wil zeggen dat het positief moet bijdragen aan het organisatiedoel. Alleen dan is de noodzakelijke managementsupport te verkrijgen (voldoende middelen, volledige medewerking van betrokkenen).
- Gebruik van een ontwikkelmethode.

Faalfactoren

- Slechte prijs/performance van shells. Goedkope shells zijn te gelimiteerd voor gebruik in een produktie-omgeving. Dure shells hebben over het algemeen veel te veel mogelijkheden.
- Geen bereidheid tot acceptatie door betrokkenen. Redenen: geen vertrouwen in de regelbank (zeker indien die niet in-house is ontwikkeld), onvoldoende betrokken bij het ontwikkelproces (veel methoden richten zich alleen op de expert en hebben niet of nauwelijks aandacht voor de gebruiker).
- Oppervlakkigheid van de kennis in het systeem.
- Onvoldoende marketing van het systeem, indien dit uit een research-omgeving komt. Dit uit zich o.a. in: te weinig aandacht voor de mens-machine interface, documentatie en gebruikersondersteuning.

Een aantal van deze slaag- en faalfactoren is uitgewerkt in (Beckman, 1991). Hierin wordt ook de capaciteit van de systeemontwerper genoemd als belangrijke slaagfactor voor een kennissysteemproject.

In (Mastrigt et al, 1989) staat een onderzoek naar slaag- en faalfactoren beschreven, dat in Nederland werd uitgevoerd. De eerder genoemde slaag- en faalfactoren zijn ook in dit onderzoek naar voren gekomen. Het onderzoek gaf veel slaag- en faalfactoren, die niet alleen voor kennissystemen gelden. Zo werd als slaagfactor genoemd, dat gebruikers in een vroegtijdig stadium bij de ontwikkeling moeten worden betrokken. Dit geldt echter voor ieder systeemontwikkelproject. Slaag- en faalfactoren die uit dit onderzoek naar voren kwamen, die typisch gelden voor kennissysteem-projecten en die nog niet werden genoemd, zijn: gebruik van standaard hardware en een goede kennisoverdracht (slaagfactoren); validatie van kennis

door verschillende experts, verificatie van de kennis, tijdgebrek bij experts, subjectieve karakter van kennis, geen goed kennismodel opstellen en het formaliseren van de kennis van de expert (faalfactoren). In (Wognum en Lippolt, 1991) staat een vervolgonderzoek beschreven. Als typisch kennissysteem-aspect kwam daaruit onder meer naar voren, dat het onderhoud van een kennissysteem een belangrijke slaagfactor is. Indien dit niet goed is geregeld, veroudert de kennis in het systeem en verliest het systeem daardoor zijn waarde.

Samenvattend kan worden gesteld, dat grotere onderhoudbaarheid van de kennis inderdaad een belangrijke toegevoegde waarde is van een kennissysteem ten opzichte van een systeem met een traditionele architectuur. Wel moet tevens aandacht worden geschonken aan andere aspecten die meer op organisatorisch en managementvlak liggen. In het bijzonder lijkt integratie van een kennissysteem in de reeds aanwezige systeemstruktuur een belangrijke slaagfactor te zijn. In het volgende hoofdstuk zal hier nader op worden ingegaan.

6 INTEGRATIE

Zoals in het vorige hoofdstuk werd opgemerkt, is integratie van een kennissysteem in de bestaande systeemstruktuur een belangrijke succesfactor voor het laten slagen van een kennissysteemproject. In dit hoofdstuk zal het begrip integratie nader worden uitgewerkt. Paragraaf 6.1 beschrijft een aantal verschijningsvormen van integratie. In paragraaf 6.2 wordt een praktijkonderzoek beschreven, dat tot doel had meer inzicht te krijgen in factoren die een rol spelen bij integratie van kennissystemen. Dit onderzoek wordt in paragraaf 6.3 geëvalueerd. In paragraaf 6.4 worden tenslotte enkele aspecten van integratie beschreven.

6.1 VERSCHIJNINGSVORMEN VAN INTEGRATIE

Stand-alone kennissystemen kunnen bedoeld zijn om door één gebruiker te worden gebruikt. In deze situatie is er geen sprake van enige vorm van integratie. Een eerste vorm van integratie ontstaat als een stand-alone kennissysteem tot doel heeft dat een groep van gebruikers de taak, waarbij het systeem ondersteuning moet leveren, op een consistente wijze uitvoert. In de literatuur wordt dit meermalen als voordeel genoemd bij het gebruiken van een kennissysteem (zie bijvoorbeeld (Harmon en Maus, 1988) en (Waterman, 1986)). Dit voordeel van integratie kan verklaard worden vanuit de procesvisie op kennissystemen. Het betreft dan meestal een taak waarvoor menselijke expertise vereist is. Een voorbeeld van zo'n systeem is het beleggingsadviessysteem van Robeco[1]. Dit systeem heeft tot doel beleggingsadviseurs te ondersteunen bij het uitbrengen van een (telefonisch) advies over een te volgen beleggingsstrategie. Op deze wijze wordt bereikt, dat de adviezen consistent zijn, zowel tussen de verschillende adviseurs als in de tijd. In (De Hoog en Van der Spek, 1992a, p. 69) wordt dit ook als voordeel van het toepassen van kennissystemen genoemd.

Een andere verschijningsvorm van integratie is de situatie, waarbij twee of meer verschillende kennissystemen gebruik maken van dezelfde regelbank (integratie over kennissysteemtoepassingen heen). Deze situatie wordt ook wel aangeduid als het *gelijktijdig gebruiken van regelbanken*. In (Neches, 1992) wordt een (toekomst)visie geschetst, waarin het bouwen van een regelbank niet betekent dat "van scratch" wordt gebouwd, maar dat de bouwer in staat is regels uit eerder gebouwde regelbanken te hergebruiken danwel mede gebruik te maken van bestaande regelbanken. Deze visie maakt tevens duidelijk dat gelijktijdig gebruik van regelbanken en *hergebruik* van regels beide kunnen worden gezien als een middel om te komen tot een efficiëntere wijze van het ontwikkelen van

[1]De informatie over dit systeem is verkregen bij een lezing voor de VRI op 29 september 1992.

kennissystemen dan tot nu toe het geval is. Op deze wijze kunnen ook shells ontstaan, waarin niet alleen de redeneercomponent wordt hergebruikt, maar waarin ook de regelbank al gevuld is met domeinspecifieke regels. In paragraaf 6.4 zal hier nader op worden ingegaan.

Een derde verschijningsvorm van integratie is de situatie, waarin een kennissysteem deel uitmaakt van een groter informatiesysteem (integratie tussen kennissysteem en andere typen informatiesystemen). In de situatie, waarin sprake is van zo min mogelijk afhankelijkheid tussen de diverse systeemdelen bestaat er een ingaande en/of uitgaande datastroom van de kennissysteemmodule naar een of meer andere deelsystemen. Voorbeelden van deze vorm van systeemintegratie staan beschreven in een veelheid aan literatuur (zie bijvoorbeeld (Kwee en van den Herik (1990), pp. 59ff, 143ff, 149ff, 187ff), (Van der Spek en Treur (1991), pp. 71ff, 101ff, 151ff), (Liebowitz (1991), pp. 62ff, 522ff, 623ff, 1132ff, 1258ff, 2058ff, 3019)). In deze beschrijvingen wordt echter niet stilgestaan bij de overwegingen die gemaakt zijn om tot integratie over te gaan. De volgende paragraaf beschrijft een praktijkonderzoek dat werd uitgevoerd om hier meer inzicht in te krijgen.

6.2 CASUS BESCHRIJVINGEN

6.2.1 INLEIDING

Om inzicht te krijgen in de problematiek die in de praktijk bij integratie van een kennissysteem met een "traditioneel" informatiesysteem worden ondervonden, is bij een drietal organisaties een onderzoek uitgevoerd. Aan de hand van een applicatie waar zo'n integratie daadwerkelijk tot stand is gebracht, zijn de ervaringen in kaart gebracht. De applicaties zullen worden besproken aan de hand van de volgende punten:

- Beschrijving en doel van de applicatie. Hierin wordt het domein waarop het systeem is ingezet beschreven.
- Beschrijving van de werking en integratie. Bij de beschrijving van de integratie wordt zowel een "logische" beschrijving bedoeld (welke invoer komt vanuit welke systemen en welke uitvoer gaat naar welke systemen) als een "technische" (welk hardware platform wordt gebruikt).
- Gevolgde methode van ontwikkeling. Hierbij wordt in het bijzonder stilgestaan bij de selectie van het betreffende probleem (welke overwegingen hebben geleid tot de selectie van juist dit probleem en waarom werd moeite gedaan om de integratie tot stand te brengen).

- Leerervaringen, in het bijzonder de succes- en faalfactoren voor integratie.
- Toekomstige plannen.

6.2.2 KASSA

Bij de beschrijving van dit systeem is gebruik gemaakt van (AI&AA, 1992).

BESCHRIJVING EN DOEL

De Nederlandse Belastingdienst onderscheidt onder meer de volgende typen belasting: Inkomstenbelasting (IB), vermogensbelasting (VB), omzetbelasting (OB), vennootschapsbelasting (VPB) en loonbelasting (LB). Een belastingplichtige in Nederland kan aangifte doen voor een of meer van die typen belasting, afhankelijk van zijn situatie. Zo moet een eigenaar van een eenmanszaak zowel aangifte doen van zijn omzetbelasting (meestal per kwartaal) als van zijn inkomstenbelasting (meestal per jaar). In het verleden werden de verschillende aangiften apart bekeken en beoordeeld. Deze situatie was echter niet optimaal. Aangiften kunnen eigenlijk niet los van elkaar worden beschouwd. Zo moet de behaalde omzet in een eenmanszaak in verhouding staan tot de inkomsten, die de eigenaar van de zaak over dat jaar aangeeft. Verder kan zo de situatie ontstaan dat een belastingplichtige voor iedere aangifte apart controle kan verwachten waarbij een groot deel van de controles overlappend zijn.

Om deze situatie te verbeteren kent de Belastingdienst sinds enkele jaren een geïntegreerde benadering. Hiervoor werd het begrip *belastingeenheid* ingevoerd. Dit is een natuurlijk of rechtspersoon, die belastingplichtig is voor een of meer van de hiervoor genoemde belastingtypen. Aangiften worden nu gezamenlijk per belastingeenheid afgehandeld. Een belangrijke stap in dit afhandelingsproces is de *selectie*. Dit is de activiteit, waarin naar aanleiding van de aangiften wordt bekeken, welke vervolgstap moet worden uitgevoerd. Hierbij zijn er drie mogelijkheden:

- Geautomatiseerd afdoen. De aangiften zijn voldoende gespecificeerd en correct. De definitieve aanslagen kunnen worden vastgesteld conform de geldende regels.
- Kantoortoets uitvoeren. Er zijn een aantal onduidelijkheden vastgesteld, waardoor het nog niet mogelijk is om een definitieve aanslag vast te stellen. Een kantoortoets kan bijvoorbeeld bestaan uit het vergelijken van een aangifte met die van voorgaande jaren.
- Veldonderzoek uitvoeren. Op grond van de aangiften is het nodig om bij de belastingeenheid de boekhouding nader te bestuderen. Hierbij wordt onder-

scheid gemaakt tussen een velddeelonderzoek (bijvoorbeeld slechts gericht op de aangifte loonbelasting) of een volledig veldonderzoek.

Via het uitvoeren van selecties wordt ernaar gestreefd om een efficiënte en effectieve inzet van beperkte capaciteit te bereiken.

Het systeem KASSA (een acroniem voor Kennis Applicatie Signalering & Selectie van Aangiften) maakt een automatische selectie van aangiften mogelijk. De belangrijkste doelstellingen van KASSA zijn het opdoen van ervaring met geautomatiseerd selecteren en het mede vorm geven aan het selectieproces.

WERKING EN INTEGRATIE

KASSA verzamelt de benodigde dossiers. Deze zijn voor een deel reeds geautomatiseerd vastgelegd in systemen. Voor de benodigde IB-gegevens is een apart invoersysteem gemaakt, omdat de IB-gegevens nog niet geautomatiseerd worden verwerkt. Het systeem doorloopt enkele honderden selectiecriteria, die in de vorm van kennisregels zijn vastgelegd. Deze regels zijn afkomstig van een aantal selectiedeskundigen. Per dossier wordt zo een behandeladvies opgesteld. Dit advies is een van de drie mogelijke uitkomsten, zoals die in het voorgaande zijn beschreven (waarbij ook het onderscheid tussen velddeeltoets en volledige veldtoets wordt gemaakt). Indien de kennis van het systeem ontoereikend is om tot een advies te komen, geeft het als antwoord "Nader te bekijken". Indien het advies een kantoortoets of veldtoets is, geeft het systeem tevens aandachtspunten in de aangiften weer die nader moeten worden bekeken. Hierbij wordt er door het systeem rekening mee gehouden, dat alle te behandelen dossiers met de opgegeven menscapaciteit ook daadwerkelijk kunnen worden afgehandeld. Voor het uitvoeren van veldtoetsen zijn hoger gekwalificeerde mensen nodig. Indien er teveel veldtoetsen zouden moeten worden uitgevoerd, "besluit" het systeem met behulp van een aantal kennisregels een aantal adviezen van die vorm te wijzigen in een (minder intensieve en qua niveau lager gekwalificeerde) "kantoortoets".

De invoer van het systeem is afkomstig uit het centrale OB- en LB-systeem. Dit zijn produktiesystemen, die in een DB2-database op een IBM 3090 opgeslagen zijn. De VPB-gegevens zijn afkomstig van het systeem KA-VPB, die draait op een DPS-6 van Bull. Het systeem KASSA is gerealiseerd met behulp van de ADS-software. Dit is een op regels gebaseerde kennissysteemshell, waarmee voorwaarts en achterwaarts redeneren mogelijk is. Het huidige systeem bevat ongeveer 600 regels. De gegevens uit de database kunnen worden ingelezen in een Object-hiërarchie. Op dit moment bestaat deze uit ongeveer 60 classes. Door de regels in de regelbank te groeperen naar de typen belasting ontstaat een struktuur in de

regelbank. Verder bevat de regelbank regels, die kennis over de dossierstruktuur van de belastingeenheid te representeren.

Het systeem is geïnstalleerd op een IBM PS/2, model 95. Deze is aangesloten op een Token Ring netwerk. De OB- en LB-gegevens komen via de DPS-6 machine ter beschikking van KASSA. Nadat alle gegevens zijn ingelezen worden lokaal op de PS/2 de gegevens opgeslagen in DBASE-bestanden. Hierna kan het eigenlijke selectieproces beginnen. KASSA maakt tijdens het selectieproces dus niet on-line gebruik van de gegevens van de aangiften.

Het selectieproces vindt 1 keer per maand plaats. Het is een batchproces, waarbij per keer 1200 beoordelingen worden gemaakt in ongeveer 3,5 uur. Dit is gemiddeld 10,5 seconde per beoordeling. In de oude (handmatige) situatie vergde het maken van een beoordeling sec (dus zonder het bijeen zoeken van de benodigde gegevens) gemiddeld 7 minuten. De resultaten van het selectieproces worden in de vorm van een print-out ter beschikking gesteld aan de gebruiker.

METHODE VAN ONTWIKKELING

Bij de afweging om juist het selectieproces te ondersteunen met behulp van een kennissysteem hebben een aantal punten meegespeeld. Allereerst is een goed selectieproces van groot belang voor de efficiëntie en effectiviteit van de diverse diensten. Een goed selectieproces is een nodige voorwaarde voor een optimale belastingopbrengst binnen de beperkingen in (vooral) menscapaciteit. Het selectieproces paste als type taak binnen de problemen die geschikt zijn om door een kennissysteem te kunnen worden opgelost. Een kennissysteem wordt namelijk omschreven als een systeem, dat de kennis en mogelijkheden in zich heeft om taken uit te voeren op het niveau van menselijke deskundigen. Verder wordt een uniform selectiebeleid over alle diensten van belang geacht. Omdat de kennis over het selectieproces voornamelijk berust op ervaring, zou een kennissysteem deze uniformiteit moeten garanderen. Tenslotte is de kennis voortdurend in ontwikkeling (bijvoorbeeld door wijzigingen in de belastingwetgeving). Een kennissysteem geeft door zijn architectuur de hiervoor gewenste onderhoudsflexibiliteit.

Er is allereerst een prototype ontwikkeld, dat zich primair richtte op de selectie van VPB aangiften. Er werd van 400 belastingeenheden een automatische selectie gemaakt. De resultaten werden vergeleken met de handmatige selectie. De vergelijking was bevredigend. Hiermee was de haalbaarheid aangetoond van het gebruik van kennistechnologie voor selectie. Wel leverde de test op, dat het gebruikte tool niet geschikt was voor het te realiseren systeem omdat alleen voorwaarts redeneren mogelijk was.

Het huidige systeem werd methodisch ontwikkeld, d.w.z. dat allereerst een analyse plaatsvond van de gebruikte kennis en gegevens. Deze analyse leverde na 7 maanden een eerste functionele specificatie van het systeem op. Nadat met de bouw van het systeem was gestart, werd de analyse voortgezet om de resterende functionaliteit te achterhalen. Door de Object-hiërarchie in de gegevensstruktuur werd voldoende flexibiliteit verkregen om de resultaten van de tweede analyse in het systeem in te passen. De integratie met de diverse in gebruik zijnde systemen was noodzakelijk vanwege de grote hoeveelheid gegevens, die bij het selectieproces wordt gebruikt. Versie 1.0 van KASSA werd 1 januari 1992 operationeel binnen twee eenheden.

LEERERVARINGEN

Door de gebruikte methode van ontwikkelen en de flexibiliteit van de tool, om tijdens de bouw nog analyse-resultaten eenvoudig toe te kunnen voegen, zodat bouw en analyse deels parallel konden worden uitgevoerd, werd de beoogde planning gehaald. ADS werd ervaren als een geavanceerde tool. Dit geeft, na het doorlopen van een leercurve, de mogelijkheid om relatief snel een relatief complex systeem te bouwen. De tijdwinst, die bij het bouwen ontstaat, moet echter worden gebruikt voor uitgebreidere testen.

Als *succesfactoren* werden genoemd:

- De integratie is cruciaal voor de acceptatie van het systeem. Het is onacceptabel om de veelheid aan gegevens handmatig in te geven.
- Het selectiesysteem moet flexibel van opzet zijn. Hoewel het selectiebeleid zoveel mogelijk uniform moet zijn, ontstaan door concentratie van beroepstakken in bepaalde delen van het land toch regionale verschillen in de beoordeling. Zo heeft een varkensfokker in De Peel met grote waarschijnlijkheid een groter bedrijf dan een elders in het land, hetgeen een verschillend selectie-resultaat tot gevolg kan hebben. Verder kent de Belastingdienst regelmatig speciale "acties". De "Actie Schuimkraag", waarbij veel aandacht werd gegeven aan aangiften vanuit de horeca-sector is daar een bekend voorbeeld van. Het systeem moet met deze afwijkingen van het normale patroon om kunnen gaan.

TOEKOMST

De ervaringen, die op dit moment worden opgedaan met versie 1.0 van KASSA, worden verwerkt in een nieuwe versie. Deze versie wordt september 1992 ingevoerd. Tevens zal in de tweede helft van 1992 een evaluatie plaatsvinden. Deze evaluatie heeft als doelstelling het vaststellen van de effectiviteit van KASSA c.q. van de gehanteerde selectiecriteria. Tevens moet worden vastgesteld welk risico met een geautomatiseerde selectie wordt gelopen. In de

verdere toekomst moeten de aangiften, die bij de selectie het resultaat "automatisch afdoen" hebben gekregen, automatisch worden doorgegeven aan het systeem dat de aanslagen genereert.

6.2.3 RAP

Bij de beschrijving van dit systeem is gebruik gemaakt van (Beckers et al, 1990) en (Beckers et al, 1991).

BESCHRIJVING EN DOEL

Om de deskundigheid met betrekking tot medisch handelen van huisartsen te bevorderen zijn door het Nederlands Huisartsen Genootschap standaarden ontwikkeld. Deze standaarden bevatten richtlijnen die houvast kunnen bieden bij het medisch handelen van de huisarts. De standaarden zijn gericht op geregeld voorkomende klachten en aandoeningen. Om te komen tot een verhoogde kwaliteit van geneeskundige zorg dient toetsing van het medisch handelen van de huisarts plaats te vinden. Zo'n toetsing houdt in dat het medisch handelen van een arts wordt vergeleken met de richtlijnen, die in een van toepassing zijnde standaard zijn vastgelegd. Uit een toetsing kan blijken dat een aantal behandelstappen uit de standaard niet zijn uitgevoerd, terwijl ook een aantal behandelstappen kunnen zijn uitgevoerd, die niet in de standaard zijn aangegeven.

Om deze toetsing te realiseren, is een praktisch hanteerbare vorm van de standaarden noodzakelijk. Voor het uitvoeren van een toetsing door de huisarts zelf is het kennissysteem RAP (Richtlijnen Automatiserings Project) ontwikkeld.

WERKING EN INTEGRATIE

Veel huisartsen maken gebruik van een HuisartsInformatieSysteem (HIS). Dit is meestal een standaardpakket. Een onderdeel van zo'n systeem is een patiëntenkaart, vastgelegd in geautomatiseerde vorm. Veel van de gegevens die nodig zijn om een toetsing uit te kunnen voeren, zijn aanwezig op zo'n patiëntenkaart. Het ligt daarom voor de hand om een koppeling te leggen tussen een HIS en het RAP ter vermijding van dubbele invoer. De integratie tussen beide systemen wordt in fasen aangepakt. Iedere volgende fase heeft een grotere mate van complexiteit ten aanzien van de integratie. De te onderscheiden fasen zijn:

1. Bijna ieder HIS is menugestuurd. Door in het menu een keuze "Toetsing" op te nemen, kan RAP worden bereikt vanuit het HIS. De menukeuze heeft tot gevolg, dat de omgeving van het HIS wordt verlaten en RAP wordt gestart. Na afloop wordt RAP

afgesloten en keert de besturing terug naar het HIS. Hier is niet echt sprake van integratie in de zin dat er een gegevensstroom ontstaat tussen beide systemen.

2. RAP kan worden aangeroepen vanuit het HIS en fungeert als invoersysteem voor het HIS voor de patiëntgegevens. Hierbij ontstaat er een gegevensstroom tussen beide systemen en is er dus sprake van integratie. Het nadeel van deze aanpak is, dat de huisarts te maken krijgt met twee verschillende gebruikersinterfaces. Dit wordt door de meeste artsen ervaren als niet gebruiksvriendelijk. In zo'n situatie zal RAP dan niet worden gebruikt.
3. Om het probleem van verschillende gebruikersinterfaces op te lossen, worden de voor het RAP benodigde gegevens in het HIS vastgelegd. Bij het maken van een toetsing werkt RAP als het ware op de achtergrond. De resultaten van RAP worden door het HIS verwerkt en gepresenteerd aan de gebruiker. Bij deze aanpak ontstaat echter een probleem dat wordt aangeduid met "uncontrolled vocabulary". De mate van detail in de gegevens die het RAP nodig heeft, wordt in het HIS voor een gedeelte vastgelegd in vrije tekst. Dit heeft tot gevolg dat of de mate van detail niet door het HIS wordt geleverd (doordat de arts het niet heeft vastgelegd) of de terminologie, die in het HIS wordt gebruikt, niet wordt begrepen door het RAP. Dit kan worden opgelost door de terminologie, die binnen het HIS wordt gebruikt om ziektebeelden vast te leggen, uit te breiden op een zodanige wijze dat de terminologie van het RAP erin past. Omdat deze terminologie vastligt in standaard codelijsten, kan dit onafhankelijk van de leveranciers van de HIS-pakketten gebeuren. Iedere leverancier draagt er zorg voor dat zijn pakket conform de standaarden werkt.
4. Het probleem van de "uncontrolled vocabulary" kan ook worden opgelost door het RAP te voorzien van een natuurlijke taal interpretator. Een andere mogelijkheid is, om iedere ingevoerde term in het HIS op de achtergrond (onzichtbaar voor de gebruiker van het HIS) te laten controleren door het RAP op bekendheid met die term.

In de fasen 3 en 4 wordt het RAP in feite ontdaan van zijn gebruikersinterface. Wat dan overblijft is een systeem, waarbij:

Feiten worden aangeboden en
Een vraag wordt aangeboden zodat
Een antwoord wordt gegeven op de vraag over die feiten

De vraag in fase 4 is dan wat betreft de controletaak steeds: "is deze term bekend?", met als mogelijke antwoorden "Ja" of "Nee". Het RAP in deze vorm heeft dus alleen de rol van

vraag-beantwoorder. Het haalt niet zelf de gegevens uit databases, maar maakt slechts gebruik van de gegevens die aangeboden zijn.

RAP is geïmplementeerd in de kennissysteemshell ROIS. Het bestaat uit drie lagen:

- Kennisbank. Kennis wordt gerepresenteerd met behulp van een semantisch netwerk.
- Subshell. Deze bevat enerzijds een omschrijving van de typen kennis uit de kennisbank (de typen relaties, die in het semantisch netwerk aanwezig zijn) en anderzijds een beschrijving van de typen taken, die door het systeem moeten worden uitgevoerd, in relatie gebracht met de typen kennis uit de kennisbank. Een taak bestaat hierbij uit een omschrijving van de soort vraag, die kan worden verwacht en het soort antwoord, dat dan moet worden gegeven. Een voorbeeld van een vraag kan zijn "Is een toetsing met deze gegevens mogelijk? Zo nee, welke gegevens moeten er dan nog worden geleverd?". Het bijbehorende antwoord kan dan zijn "Ja" of "Nee, de volgende gegevens zijn nog nodig:...".
- Systeem. Hierin staat de programmatuur, die de kennis en gegevens uit de subshell en de kennisbank kan combineren om tot een antwoord te komen.

Het RAP draait op een IBM of compatible PC, type XT of hoger. In de praktijk betekent dit, dat het RAP geen beperkingen kent ten aanzien van te gebruiken hardware.

METHODE VAN ONTWIKKELING

Het RAP werd als een stand-alone systeem ontwikkeld en op proef geïnstalleerd bij een aantal huisartsen. Na een evaluatie van de proefperiode werd besloten tot de koppeling met het HIS om dubbele invoer van gegevens te vermijden (zie ook "Leerervaringen").

Uit eerdere ervaringen was gebleken, dat het voor toekomstige gebruikers erg lastig is om de prestatie eisen van een dergelijk systeem van tevoren te laten formuleren. In het bijzonder geldt dit voor de eisen ten aanzien van de gebruikersinterface. Door de gebruiker te confronteren met het systeem, kunnen achteraf wel uitspraken van de gebruiker betreffende deze eisen worden verkregen. De te volgen methode van ontwikkeling heeft daarom een prototyping karakter.

Zoals reeds geschetst verloopt de ontwikkeling van het systeem stapsgewijs, waarbij iedere volgende versie een uitbreiding inhoudt van een vorige versie. Zo werd in een tweede versie van RAP de mogelijkheid gegeven een toetsing uit te voeren van meerdere casussen betreffende een standaard en meerdere casussen betreffende meerdere standaarden. Daarnaast zijn de standaarden nog niet uitontwikkeld. Mede om deze redenen werd besloten de ROIS-

architectuur te gebruiken als implementatiemiddel voor het systeem vanwege de flexibiliteit die door deze architectuur wordt geboden ten aanzien van onderhoud.

LEERERVARINGEN

Vanwege de onervarenheid van de meeste huisartsen met betrekking tot het gebruik van informatiesystemen in het algemeen en niet-administratieve systemen als RAP in het bijzonder zal de koppeling zodanig moeten zijn, dat een uniforme gebruikersinterface wordt gerealiseerd. Dit betekent dat naar verwachting alleen een koppeling, die in fase 3 of 4 wordt gerealiseerd acceptabel is voor de meerderheid van de toekomstige gebruikers. Verder is afwezigheid van dubbele invoer een belangrijke factor voor uiteindelijke acceptatie van het systeem.

TOEKOMST

Momenteel bevindt de integratie zich in fase 2. Voor fase 3 is voorgesteld om de classificaties, die in het HIS worden gebruikt voor de beschrijving van de casussen, uit te breiden op een zodanige wijze, dat de terminologie van het toetsingssysteem erin past.

6.2.4 GEOLOGIX

Het systeem GEOLOGIX is een van de ongeveer 30 kennissystemen, die momenteel in gebruik zijn bij Shell. Een veelvoud van dit aantal systemen is wel ontwikkeld, maar worden om diverse redenen niet gebruikt. De systemen, die wel worden gebruikt zijn alle geïntegreerd of worden op korte termijn geïntegreerd met andere systemen. Bij de beschrijving van dit systeem is gebruik gemaakt van (Kraats, 1989).

BESCHRIJVING EN DOEL

Voordat een olieveld daadwerkelijk kan worden geëxploiteerd, hebben er al talloze werkzaamheden plaatsgevonden. Een van de werkzaamheden is het doen van proefboringen. De grond, die bij zo'n boring boven komt, wordt geanalyseerd op de mogelijke aanwezigheid van een olieveld. Hierbij wordt onder meer een schets gemaakt van de lagen, die in de bodem zijn aangetroffen. De samenstelling van ieder van die lagen geeft aanwijzingen over het al dan niet aanwezig zijn van een olieveld. In een onderzoeksgebied worden zo op diverse plaatsen boringen gedaan. Uit de schetsen van de grondlagen, die zo ontstaan, wordt een schets gemaakt, waarbij een totaalbeeld van de samenstelling van de bodem in lagen wordt verkregen. Hiervoor moet men in staat zijn de overeenkomstige lagen in ieder van de boorgaten met elkaar te verbinden. Uiteindelijk moet inzicht worden verkregen of het gebied voldoende aantrekkelijk is voor exploitatie en zo ja, op welke wijze het beste kan worden geboord.

Het systeem GEOLOGIX is in staat deze werkzaamheden geautomatiseerd uit te voeren. Een van de modules binnen GEOLOGIX is een kennissysteem dat de taak heeft de overeenkomstige grondlagen uit ieder van de boringen met elkaar te verbinden. Dit kennissysteem zal hierna nader worden beschreven.

WERKING EN INTEGRATIE

De grondanalyse van een boring gebeurt door middel van een neuraal netwerk op basis van patroonherkenning. De resultaten van ieder van deze analyses vormt de invoer voor het kennissysteem. Het kennissysteem "verbindt" bij elkaar horende grondlagen en geeft als resultaat een totaaloverzicht van de samenstelling van de bodem in het proefgebied. Dit resultaat wordt vervolgens door een andere module vertaald in een 3D-model van de bodem.

Het systeem is ontwikkeld m.b.v. eigen ontwikkelde software, Knowledge Craft en Symbolix en draait op werkstations.

METHODE VAN ONTWIKKELING

De meeste grotere kennissystemen binnen Shell zijn ontstaan vanuit de research. In deze omgeving wordt voornamelijk prototyping als ontwikkelmethode gebruikt. In de niet-researchomgeving wordt een systeem modulegewijs stapsgewijs ontwikkeld. Iedere stap bestaat uit het ontwikkelen van een module, die een bepaalde systeemtaak moet uitvoeren. Hierbij is de doorlooptijd van de ontwikkeling van een module maximaal een half jaar.

Binnen Shell wordt een kennissysteem omschreven als een systeem dat in staat is om nieuwe gegevens af te leiden met behulp van logische afleidingsregels. Dit hoeft echter niet per se te betekenen dat het systeem een architectuur kent met een afzonderlijke regelbank en redeneerprocedure. Bij de meeste bij Shell in gebruik zijnde kennissystemen moet vanuit veel data een conclusie worden bereikt.

Om te bepalen of een probleem geschikt is om met een kennissysteem te worden opgelost, werden de volgende vuistregels gebruikt:

- De kennis die nodig is om het probleem op te lossen moet enigszins stabiel zijn. Het systeem zal bij niet-stabiele kennis niet worden gebruikt omdat dan de kennis constant moet worden onderhouden.
- De kennis die in het systeem zal worden gebruikt, moet al aanwezig zijn.
- De invoer van het systeem moet zodanig zijn dat het tamelijk eenvoudig door een computer kan worden gelezen. Dit sluit bijvoorbeeld afbeeldingen die nader worden geanalyseerd uit.

- Indien het probleem kan worden opgelost met traditionele programmeertalen dan heeft dat de voorkeur, omdat dat een snellere responsetijd en een relatief eenvoudige implementatie heeft.
- Er moet vraag zijn naar de oplossing van het probleem. Dit kan betekenen, dat het probleem zich vaak voordoet en/of het een mens enkele uren kost om de oplossing van het probleem te vinden.

Het probleem van het bepalen van overeenkomstige grondlagen voldoet aan meerdere van deze criteria en werd daarom geschikt geacht om met behulp van een kennissysteem op te lossen.

Integratie van het kennissysteem met anderen systemen ligt voor dit probleem voor de hand. De invoer voor het kennissysteem is afkomstig uit een geautomatiseerd systeem terwijl de uitvoer door een ander geautomatiseerd systeem wordt verwerkt.

LEERERVARINGEN

Op grond van de ervaringen, die bij Shell zijn opgedaan bij de ontwikkeling van enkele honderden kennissystemen, wordt integratie van een kennissysteem in een totale systeemstruktuur als nodige voorwaarde genoemd voor het succes van het kennissysteem. Een standalone systeem is vaak zo klein dat het slechts enkele malen zal worden gebruikt. Daarnaast zijn succes- en faalfactoren geleerd die niet slechts te maken hebben de integratie van een kennissysteem met andere systemen. Enkele van deze factoren zijn:

- Een adequate gebruikersinterface is een van de belangrijkste succesfactoren. "Adequaat" wil hier zeggen, dat het systeem zo eenvoudig mogelijk door een gebruiker moet zijn te benaderen.
- Het gebruik van een "exotische" taal heeft vaak tot gevolg, dat het systeem te traag wordt of niet kan worden gebruikt vanwege bijzondere hardware-eisen.
- Aandacht voor de gebruikerseisen (zowel de functionele als de prestatie-eisen) is essentieel voor de uiteindelijke acceptatie van het systeem.

TOEKOMST

Er zal op meerdere gebieden nader onderzoek moeten plaatsvinden. Enkele van die onderzoeken betreffen de te ontwikkelen gebruikersinterfaces, het begrijpen van teksten in natuurlijke taal en databasetechnologie voor het combineren van data uit verschillende grote databases. GEOLOGIX kan profijt hebben van de resultaten van een of meer van die onderzoeken.

6.3 EVALUATIE

Voor elk van de beschreven integratie-cases geldt, dat het kennissysteem als aparte module van het totale systeem te beschouwen is. De communicatie met de andere delen van het systeem komt tot stand, doordat het kennissysteem gegevens als invoer krijgt, afkomstig van een ander deelsysteem. De communicatie verloopt rechtstreeks (bijvoorbeeld bij de koppeling van het RAP en het HIS) of via een gemeenschappelijke database (bijvoorbeeld bij KASSA). De uitvoer van het kennissysteem wordt niet altijd gebruikt door een ander deelsysteem, c.q. in een database opgeslagen, maar wordt alleen aan de gebruiker getoond (RAP en KASSA).

De integratie kan op twee verschillende manieren tot stand komen:

1. Het kennissysteem wordt als stand-alone systeem ontwikkeld. Omdat het systeem veelal gebruik maakt van gegevens, die reeds in bestaande databases worden opgeslagen, wordt tot integratie in een grotere informatiesysteemstruktuur besloten.
2. Het kennissysteem wordt ontwikkeld als module in een groter informatiesysteem (GEOLOGIX).

Het tweede geval kan gevolgen hebben voor de methode van ontwikkelen van een informatiesysteem. Er zal niet altijd al van tevoren een duidelijk beeld zijn of er in het op te lossen probleem een of meer deelproblemen zijn die beter met behulp van kennissystemen zijn op te lossen. In (Kusters en Schuwer, 1991) wordt betoogd dat bij een procesgerichte wijze van systeemontwikkeling een functionele probleemdecompositie leidt tot een verzameling deelproblemen. Voor ieder van die deelproblemen afzonderlijk kan door het toepassen van de richtlijnen uit hoofdstuk 5, worden nagegaan of het een geschikt probleem is voor een kennissysteemoplossing. Welke wijze van systeemontwikkeling echter wordt gevolgd, een ruimere aandacht is noodzakelijk voor het opstellen van een model van de kennis die een rol speelt in het informatiesysteem. Anders gezegd: een kennistechnologische blik op systeemontwikkeling (in de zin, zoals beschreven in hoofdstuk 5) leidt op een natuurlijke wijze tot informatiesystemen waar kennissysteem-modules een geïntegreerd deel van uitmaken.

In alle drie cases werd om efficiencyredenen besloten om het kennissysteem te integreren in een grotere systeemarchitectuur. Door de integratie wordt vermeden dat gegevens meer dan een keer moeten worden ingevoerd. Deze efficiencywinst wordt door de gebruikers zo belangrijk gevonden dat het een kritieke succesfactor is voor het daadwerkelijk gebruiken van een kennissysteem. Ook uit de literatuur die aan het einde van paragraaf 6.1 werd genoemd,

blijken efficiency overwegingen geleid te hebben tot de beslissing om tot integratie over te gaan, hoewel dit nergens expliciet zo werd genoemd.

6.4 ASPECTEN VAN INTEGRATIE

Bij de diverse vormen van integratie kan men zich afvragen of een extra toegevoegde waarde wordt bereikt door een systeemarchitectuur te gebruiken zoals in het XYFAR-model staat beschreven.

Doordat bij de eerste vorm van integratie (het gebruik van een stand-alone kennissysteem door meerdere gebruikers) wordt uitgegaan van een procesvisie op kennissystemen is in hoofdstuk 5 reeds opgemerkt dat het voor dit geval onduidelijk is welke toegevoegde waarde een systeem met een architectuur uit het XYFAR-model heeft.

De toegevoegde waarde van een XYFAR-architectuur voor de vorm van integratie waarbij een kennissysteem een deelsysteem is van een groter informatiesysteem, wijkt niet af van die van een stand-alone kennissysteem zoals in hoofdstuk 5 reeds beschreven. Bij KASSA en RAP was die toegevoegde waarde een van de redenen om voor zo'n architectuur te kiezen.

Bij de vorm van integratie waarbij gelijktijdig gebruik maken van een regelbank aan de orde is, kan een kennissysteem conceptueel goed worden beschreven met behulp van het XYFAR-model. Beschouw hierbij dat deel van de regelbank dat door een specifiek kennissysteem wordt gebruikt, als *de* regelbank voor dat kennissysteem. Het redeneerdeel van een kennissysteem geeft in feite aan op welke wijze de regelbank door het systeem zal worden gebruikt. Veelal zal dit betekenen, dat het redeneerdeel zodanig is dat het systeem voor de taak die het moet uitvoeren, de meest efficiënte wijze van redeneren zal gebruiken. Zoals in bijlage A bewezen, hangt de uitkomst van een redenering niet af van de inhoud van het redeneerdeel (behoudens de externe regel). Hoewel verschillende kennissystemen van dezelfde regelbank gebruik maken, hoeft dit niet te betekenen dat de redeneerdelen hetzelfde zijn. Zo kan bij de situatie met twee kennissystemen het ene kennissysteem uitgaan van veel data om een taak uit te voeren, terwijl de andere juist weinig data gebruikt. Het eerstgenoemde kennissysteem heeft dan baat bij een backward chaining redeneerstrategie, terwijl het laatstgenoemde systeem juist een forward chaining strategie kan gebruiken. Dit pleit ervoor om in zo'n situatie het redeneerdeel bij ieder kennissysteem afzonderlijk te formuleren. Vanwege de eis dat de componenten van een kennissysteem afzonderlijk benaderbaar moeten zijn, hoeft dit niet tot problemen te leiden bij veranderingen aan

regelbank en/of redeneerdeel. Sterker: indien bij modificatie van de (gelijktijdig gebruikte) regelbank ten behoeve van een kennissysteem de werking van de andere kennissystemen die tevens van de regelbank gebruik maken, onbedoeld wordt beïnvloed, dan is er bijna zeker sprake van het opnemen van redeneeraspecten in de regelbank of het opnemen van regelbank-aspecten in het redeneerdeel. Dit betekent wel dat bij de situatie van een gelijktijdig gebruikte regelbank, de ontwerper tijdens het ontwerpproces zich nog meer dan in een andere situatie rekenschap moet geven van welke kennisaspecten in de regelbank en welke aspecten in het redeneerdeel horen. Het raamwerk dat door het XYFAR-model wordt geboden, kan hierbij een goede leidraad zijn.

Extra voordelen van een gelijktijdig gebruikte regelbank (naast de voordelen, die reeds zijn genoemd in hoofdstuk 5) zijn analoog aan die van een gelijktijdig gebruikte database (Everest, 1986). Deze zijn:

- *Minimale redundantie van regels.* Idealiter zal iedere regel slechts een keer worden vastgelegd. Indien hiervan (om implementatieredenen) wordt afgeweken dan is de redundantie die dan ontstaat, onder controle.
- *Consistentie van regels.* Indien regels op meerdere plaatsen zijn opgeslagen, niet uitgaande van een logisch totaalbeeld, ontstaat het gevaar van inconsistentie. Wijzigingen aan een regel hoeven dan niet meer in alle occurrences worden doorgevoerd waardoor verschillende versies van eenzelfde regel kunnen ontstaan.
- *Mogelijkheid tot regelmanagement* (vergelijkbaar met datamanagement). Juist door het grotere overzicht dat een gelijktijdig gebruikte regelbank kan bieden, wordt het uitoefenen van deze functie beter mogelijk. Onder meer kan hierbij worden gedacht aan standaardisatie van regels (bijvoorbeeld in de naamgeving), toekennen van bevoegdheden voor het gebruik van de regels en het definiëren van user views op de regelbank.

Typische problemen die bij databases voortkomen door gemeenschappelijk gebruik van de data (zoals deadlock en lost update), komen niet of in zeer beperkte mate voor bij gebruik van een gelijktijdig gebruikte regelbank. Dit komt voort uit het feit dat de genoemde problemen samenhangen met het wijzigen van de inhoud van de database (toevoegen, verwijderen of wijzigen van records). Toevoegen, wijzigen of verwijderen van regels uit een regelbank gebeurt echter veel minder frequent dan het wijzigen van de inhoud van een database. Het gelijktijdig gebruiken van een regelbank gebeurt dan ook nagenoeg alleen in de vorm van het raadplegen van regels voor het uitvoeren van een redenering.

Tenslotte kan door het gebruik van een gelijktijdig gebruikte regelbank de situatie ontstaan, dat een kennissysteem gegevens afleidt waarvoor de gebruiker van dat systeem geen raadpleegbevoegdheid heeft. Deze situatie ontstaat indien de regelbank onvoldoende beschermd wordt tegen onbevoegd raadplegen. Deze security-eis moet worden vervuld door de programmatuur. Een KBMS zoals in hoofdstuk 5 omschreven, zou deze functionaliteit moeten bieden.

In (Neches, 1992) wordt een visie gegeven op de wijze, waarop hergebruik van regelbanken en probleemspecifieke redeneermechanismen kan worden gerealiseerd. Het systeem wordt beschouwd als een produkt dat wordt samengesteld uit standaard modules die "off the shelf" aanwezig zijn, maar die uitgebreid kunnen worden met applicatiespecifieke redeneerstrategieën. Dit systeem redeneert met behulp van data en regels die afkomstig zijn uit deels bestaande en deels nieuwe databases c.q. regelbanken. Behalve de systeemcomponenten, die in het XYFAR-model worden onderscheiden, kan een systeem ook meer conventionele modules bevatten zoals een spreadsheet of E-mail. Voor de communicatie tussen (interne) kennisbanken of tussen het systeem en externe applicaties wordt een standaardtaal voorgesteld (KQML: Knowledge Query and Manipulation Language) die eenzelfde rol vervult in de communicatie als SQL momenteel heeft voor communicatie met conventionele databases. De diverse componenten zijn opgeslagen in diverse bibliotheken. Op deze wijze kan het maken van een kennissysteem worden beschouwd als een assemblageproces dat deels kan worden gekarakteriseerd als assemble-to-order (voor het standaarddeel) en deels als engineer-to-order (voor het applicatiespecifieke deel). Deze visie is daarmee in lijn met de ideeën over een "softwarefabriek" zoals die staan beschreven in (Van Genuchten, 1991).

Hoewel de visie, zoals die in het artikel van Neches wordt ontvouwd voor een praktische bruikbaarheid nog toekomstmuziek is, mag worden verwacht dat behoefte zal ontstaan aan de mogelijkheden voor gelijktijdig gebruik en hergebruik van regelbanken. Behalve de eerder geschetste voor de hand liggende *efficiency*voordelen, die van een dergelijke aanpak mag worden verwacht kan er ook een *strategische* betekenis worden gehecht aan het op die wijze ontstaan van grote kennisbanken die gemeenschappelijk gebruikte kennis bevatten. Hoewel ieder bedrijf op dit moment zal beschikken over een stuk gemeenschappelijke kennis, is het nog erg moeilijk deze kennis ook daadwerkelijk te gebruiken. Dit is onder meer te wijten aan de wijze van vastlegging van die kennis (in de vorm van handboeken en procedures) en mede daardoor ook aan het ontbreken van inzicht in de aanwezige kennis. Het daadwerkelijk inzicht hebben in en gebruiken van een kern van voor een bedrijf essentiële kennis leidt onder meer tot een betere dienstverlening naar de markt ((De Hoog en Van der Spek (1992a), p. 69), (De Hoog en Van der Spek (1992b), p. 174)). Het Robeco-systeem is daar door de

consistente advisering richting adviesvragers, een voorbeeld van. Omdat voor veel bedrijven een optimale dienstverlening naar de markt een kritieke succesfactor is om te overleven (gegeven de situatie van een buyers market) bepaalt de aanwezige infrastruktuur van kennis mede de kracht van een organisatie. Dit alleen is voldoende reden voor iedere organisatie om ontwikkelingen te starten die dit inzicht in de gemeenschappelijke kennis moet vergroten.

7 RESULTATEN EN AANBEVELINGEN

In dit afsluitende hoofdstuk zullen in paragraaf 7.1 de resultaten van deze studie worden geformuleerd. In paragraaf 7.2 zullen enkele vraagstukken worden geformuleerd, die het onderwerp kunnen zijn van verder onderzoek.

7.1 RESULTATEN

Het doel van deze studie was te komen tot ontwerprichtlijnen, die gebruikt kunnen worden om te bepalen of het in een specifieke situatie een verstandige keuze is, van een kennissysteem gebruik te maken. Hiervoor werd allereerst een definitie ontwikkeld van een kennissysteem, dat bij het ontwerpen van de richtlijnen als uitgangspunt diende. Definities van kennissystemen in de literatuur kunnen worden geclassificeerd naar het accent, dat daarin wordt gelegd op

- ofwel de aanwezigheid van een model van menselijke expertise
- ofwel de aanwezigheid van een zekere architectuur binnen een kennissysteem.

Bij deze studie is uitgegaan van de laatste klasse van definities. Omdat geen van de gevonden definities binnen deze klasse voldoende nauwkeurig was om te gebruiken als uitgangspunt binnen deze studie, werd een eigen definitie ontwikkeld in de vorm van een formeel architectuurmodel van een kennissysteem (het XYFAR-model). In dit model wordt de database, de regelbank en een redeneerprocedure als de kern van een kennissysteem aangeduid. De werking van een kennissysteem met deze architectuur staat grotendeels formeel beschreven in (Lloyd, 1987). Het model lijkt rijk genoeg om als definitie voor een kennissysteem gebruikt te kunnen worden. Het voordeel van deze definitie is de eenduidigheid ervan.

Als een afgeleid resultaat is tevens aangegeven, hoe de ontstane definitie gebruikt kan worden als richtlijn bij het conceptueel specificeren van kennissystemen en het onderling vergelijken van software hulpmiddelen voor de implementatie van kennissystemen. Hierbij lijkt het model uit de definitie meer instrumenteel en daardoor dichter bij een implementatiemodel te liggen dan een soortgelijk model uit de methode DESIRE.

De in deze studie gegeven definitie van een kennissysteem geeft aanwijzingen voor het herkennen van die situaties waar kennissystemen een geschikte wijze van probleemoplossen zouden kunnen zijn. Bij dit herkennen van probleemsituaties moet onderscheid worden gemaakt tussen geschiktheid voor een kennistechnologische visie op systeemontwikkeling en

geschiktheid voor het ontwikkelen van een kennissysteem. Richtlijnen voor het herkennen van probleemsituaties waarbij een kennistechnologische visie op systeemontwikkeling nuttig is, zijn uit de literatuur te halen. In de literatuur wordt daarbij vaak geen onderscheid gemaakt tussen geschiktheid voor kennistechnologie en geschiktheid voor een kennissysteemoplossing. Bij het herkennen van probleemsituaties moet eerst worden nagegaan of het probleem geschikt is voor een kennistechnologische visie op systeemontwikkeling. Indien dit het geval is, kan geschiktheid voor het gebruiken van een kennissysteem als oplossing worden nagegaan.

Twee stappenplannen uit de literatuur (resp (Waterman, 1986) en (Kolman et al, 1992)) kunnen worden samengevoegd om geschiktheid voor een kennistechnologische aanpak van systeemontwikkeling te herkennen. Andere indelingen blijven ofwel te vaag (bijvoorbeeld (Chorafas, 1987)), ofwel komen overeen met het stappenplan van Waterman (bijvoorbeeld (Prerau, 1990)), ofwel zijn een beschouwingsniveau hoger en daardoor minder concreet (bijvoorbeeld de probleemtypologieën uit (Breuker et al, 1987) en (Martin en Law, 1988)). Deze observaties, tesamen met de in hoofdstuk 5 gedane observatie dat de twee stappenplannen nauwelijks overlap vertonen, suggereren dat de verzameling kenmerken uit de twee stappenplannen voldoende volledig zijn. Na samenvoeging van de twee stappenplannen ontstaat zo de volgende verzameling ontwerprichtlijnen voor het herkennen van geschiktheid voor toepassing van kennistechnologie. Hierbij wordt de indeling van Kolman et al als primaire ingang gebruikt. De indeling van Waterman (veralgemeniseerd naar "kennistechnologie" in plaats van "kennissysteem") wordt als secundaire ingang gebruikt. Alleen die kenmerken worden genoemd, waarbij van tevoren kan worden vastgesteld of ze in een specifieke situatie geldig zijn (eventueel na het uitvoeren van een pilot-studie).

Voor ieder kenmerk in de tabel geldt dat de mate, waarin het kenmerk geldig is, evenredig is met de geschiktheid voor toepassing van kennistechnologie. Het belangrijkste resultaat van zo'n aanpak is een model van de kennis. Dit model heeft tot gevolg, dat het op te lossen probleem inzichtelijker wordt.

OMSTANDIGHEDEN	
Mogelijk	Indien gebruik moet worden gemaakt van een of meer experts Er bestaat een erkende expert Experts kunnen oplossing verwoorden Taak is niet te moeilijk Taak is niet te makkelijk Taak is niet slecht begrepen
Gerechtvaardigd	Oplossing van de taak geeft een hoog rendement Kennis dreigt verloren te gaan Kennis is schaars Kennis wordt gedistribueerd Kennis wordt gebundeld Kennis is moeilijk bereikbaar Snelheid van interpretatie is belangrijk
Zinvol	Gebruiker en bron van kennis van verschillend niveau Gebruikers hebben onderling een ongelijke hoeveelheid ervaring
FUNCTIES	
Mogelijk	-
Gerechtvaardigd	Betrouwbaarheid van het gebruik van kennis moet worden vergroot
Zinvol	Er is sprake van het afleiden van kennis uit gegevens Begeleiding nodig bij gebruik van kennis door gebruiker Gebruik van kennis moet worden begrensd Gebruik van kennis moet worden geregistreerd Kennis moet worden geformaliseerd Kennis moet toegankelijk worden gemaakt
TYPE KENNIS	
Mogelijk	Er is geen gezond verstand kennis nodig Er is alleen cognitieve vaardigheid nodig
Gerechtvaardigd	-
Zinvol	Kennis berust op ervaring Kennis is niet volledig

Tabel 7.1 Kenmerken die een kennistechnologische benadering indiceren

De volgende kenmerken geven daarenboven aan of het zinvol is om een kennissysteem te gebruiken als wijze van oplossing.

- Er is behoefte aan het geven van uitleg over het resultaat of de gevolgde redenering
- Er is behoefte aan het gebruiken van een kennisbank door meerdere applicaties
- Er is behoefte aan het hergebruiken van een regelbank of redeneerprocedure
- Er is een groot aantal regels en/of metaregels
- De verzameling regels en/of metaregels is complex
- De verzameling regels en/of metaregels verandert regelmatig

Het grote voordeel dat een kennissysteem biedt ten opzichte van een conventioneel systeem is de grotere onderhoudbaarheid van de bij het probleem gebruikte kennis. In die zin is er een grote analogie met het ontstaan van en de voordelen die DBMS'en bieden in situaties, waar dezelfde kenmerken (grootte, complexiteit en veel wijzigingen) gelden voor de bij een probleem gebruikte verzameling gegevens.

Indien de kenmerken die in tabel 7.1 worden genoemd niet of niet in voldoende mate aanwezig zijn, geeft een kennistechnologische aanpak geen hoge toegevoegde waarde ten opzichte van een traditionele aanpak. Analoog geldt dit voor de afwezigheid van de kenmerken die een kennissysteem-oplossing indiceren. Daarnaast geldt ook dat een hoge response-eis moeilijk te realiseren lijkt met een architectuur van een kennissysteem. In dat geval zal een afweging moeten worden gemaakt welke eisen het belangrijkste zijn in een specifieke situatie.

In veel situaties zullen de gegevens waar het kennissysteem gebruik van maakt, reeds in geautomatiseerde bestanden vastliggen. Een nodige eis voor het daadwerkelijk gebruiken van het systeem is dat het systeem daadwerkelijk gebruik kan maken van die gegevens. Deze efficiency eis heeft tot gevolg dat een kennissysteem geïntegreerd wordt in de bestaande systeeminfrastruktuur. Het is echter denkbaar dat kennissystemen op den duur zullen dienen als middel van vastlegging van een voor het functioneren van een organisatie primair noodzakelijke kennis, zodanig dat iedere applicatie er toegang tot heeft. De toenemende noodzaak voor organisaties om zich klantgericht op te stellen in een buyers market is de reden voor de strategische plaats die een (geïntegreerd) kennissysteem dan heeft.

7.2 AANBEVELINGEN

Deze studie heeft het belang van kennissystemen binnen de totale informatievoorziening in een organisatie duidelijk gemaakt. Er blijven echter vragen onbeantwoord c.q. enkele vragen zijn door deze studie ontstaan. Op een aantal daarvan zal kort worden ingegaan.

In deze studie is aangegeven hoe software hulpmiddelen onderling kunnen worden vergeleken op het aspect "aanwezigheid van de in het XYFAR-model onderscheiden componenten". Er is al aangegeven dat ook andere aspecten een rol spelen bij de aanschaf van deze hulpmiddelen (prijs, inpasbaarheid in een bestaande hardware en software architectuur). Verder onderzoek zou alle aspecten die van invloed zijn op de aanschaf van dit type hulpmiddelen, op een rij moeten krijgen. Hierbij kan worden voortgebouwd op beslisfactoren voor conventionele software. Nadat de beslisfactoren helder zijn, kan een case worden ontwikkeld met het doel die te gebruiken bij het daadwerkelijk testen en onderling vergelijken van software hulpmiddelen voor de implementatie van kennissystemen. Zo'n case is te vergelijken met de zgn. "Ziekenhuisdatabase" (De Brock, 1990). Deze database voor een (denkbeeldig) ziekenhuis wordt al enkele jaren gebruikt voor het testen en vergelijken van RDBMS'en.

De visie op het herkennen van geschikte probleemgebieden tijdens een procesgerichte systeemontwikkeling, zoals beschreven staat in (Kusters en Schuwer, 1991) is op puur theoretische overwegingen tot stand gekomen. Er dient nog te worden onderzocht, of de geschetste aanpak ook praktische waarde heeft. Tevens moet nog worden nagegaan of de resultaten ook gelden indien een gegevensgerichte of objectgerichte aanpak van systeemontwikkeling wordt gevolgd. Hierbij kan dan tevens worden aangegeven voor welke stappen welke technieken als beschrijvingswijze het meest geschikt zijn.

In deze studie is geen aandacht besteed aan het migratiepad dat een organisatie moet afleggen om te komen tot een situatie waarbij de primair noodzakelijke kennis vastligt in kennisbanken in plaats van in de programmatuur. Een kennisverwervingstraject in zo'n geval zal bestaan uit het analyseren van de aanwezige programmatuur om een model van de kennis, die daarin verstopt is, te verkrijgen. Deze aanpak kent echter een aantal problemen:

- De kennis is gebaseerd op de toestand zoals die tijdens de bouw van het programma gold. Dit hoeft echter niet de huidige kennis te zijn
- De redeneerkennis is moeilijk uit de code te achterhalen. Dit komt onder meer, doordat diverse vertaalslagen zullen worden gemaakt vanuit een logisch model van redeneren naar een efficiënt programma. Menselijk redeneren (met het doel om

efficiënt een resultaat te bereiken) hoeft echter niet hetzelfde te zijn als efficiënt machinaal redeneren.

- Er kunnen inconsistenties in de programmatuur optreden (bijvoorbeeld als gevolg van homoniemen en synoniemen)

Naast programma-analyse zal derhalve ook een aanvullende kennisverwerving moeten plaatsvinden om de genoemde problemen aan te pakken. Een nadere studie zal moeten uitwijzen of en wanneer het voor organisaties zinvol is om op deze wijze te komen tot een model van de (aanwezige) kennis.

In hoofdstuk 3 werd een aanbeveling gegeven voor het combineren van KADS, DESIRE en het XYFAR-model om te komen tot een specificatie van een kennissysteem. Zeker bij een meer geïntegreerde ontwikkeling van kennissystemen lijkt het ook raadzaam de toepasbaarheid van traditionele methoden na te gaan voor het ontwikkelen van kennissystemen.

De ideeën in dit boek indiceren, dat praktische toepasbaarheid van kennissystemen haalbaar lijkt. Hierbij is het van groot belang dat de goede toepassingen geselecteerd worden. De richtlijnen uit hoofdstuk 5 zijn voor het bepalen van toepasbaarheid een hulpmiddel. Het inzicht in de mogelijkheden van deze technologie, zoals die uit deze studie naar voren zijn gekomen, geeft daarnaast aanwijzingen voor een meer strategische plaats van kennissystemen binnen een organisatie. Dit zal uiteindelijk opleveren, dat denken over en omgaan met het vastleggen van kennis bij de ontwikkeling van applicaties net zo vanzelfsprekend is als dat nu is voor gegevens. Uiteindelijk zal dit het inzicht in informatiesystemen en hun werking vergroten.

BIJLAGEN

BIJLAGE A EEN FORMELE DEFINITIE VAN EEN KENNISSYSTEEM

In deze bijlage zal een formele definitie van een kennissysteem worden opgebouwd. Hierbij zal worden uitgegaan van de resultaten uit (Eiben en Schuwer, 1991) en (Lloyd, 1987).

De definitie van een kennissysteem zal zich beperken tot alleen de kern van een kennissysteem. Deze bestaat uit een database, een regelbank en een redeneermechanisme. De definitie zal ieder van die componenten definiëren en tevens de verbanden aangeven tussen de componenten.

De definitie zal als volgt worden opgebouwd. Allereerst worden in paragraaf A.1 enkele basisbegrippen geïntroduceerd. Vervolgens wordt in paragraaf A.2, uitgaande van de basisbegrippen, het "statische" deel van een kennissysteem beschreven. Dit zal gebeuren door het opbouwen en definiëren van het begrip "kennismodel". Paragraaf A.3 definieert het "dynamische" redeneergedeelte (waarbij de redeneringen zullen plaatsvinden, gegeven een kennismodel). In paragraaf A.4 wordt de definitie van een kennissysteem gegeven, voortbouwend op het redeneermodel, dat in paragraaf A.3 is opgebouwd.

Het begin van een notatie-afspraak, definitie, voorbeeld of stelling wordt aangegeven door **NOTATIE**, **DEFINITIE**, **VOORBEELD** of **STELLING**. Het einde zal steeds worden aangeduid met behulp van het symbool □.

NOTATIE
Als A een verzameling is, dan wordt met $\wp A$ de machtsverzameling van A aangeduid:

$$\wp A = \{ B \mid B \subseteq A \}$$

(d.i. de verzameling van alle deelverzamelingen van A).
□

A.1 BASISBEGRIPPEN

Allereerst worden de taalelementen gedefinieerd, waarmee de formele definitie zal worden opgebouwd. Bij het geven van de definities zal gebruik worden gemaakt van eerste orde logica (onder meer als taal).

De meest primitieve taalelementen, die gebruikt worden, zijn functiesymbolen, relatiesymbolen, constanten en variabelen. Daarover zullen allereerst notatie-afspraken worden gemaakt.

Voor een *functiesymbool* wordt meestal de letter f of g (eventueel met een index) gebruikt. Voor een *relatiesymbool* (synoniem: predicaatsymbool) wordt meestal de letter p of q gebruikt. Een relatiesymbool is in feite een bijzonder functiesymbool: in het waardenbereik van een relatiesymbool komen alleen logische waarden (TRUE, FALSE en UNDEF) voor. Bij een functie- en een relatiesymbool hoort ook te worden aangegeven, hoeveel argumenten dit symbool heeft. Het aantal argumenten van een functie- of relatiesymbool heet de *ariteit* van dat symbool. Een *constante* is een functiesymbool met ariteit 0. Voor constanten worden meestal de letters a,b,c,.. gebruikt. Een *variabele* is een symbool, waarvoor (afhankelijk van de context, waarin de variabele wordt gebruikt) constanten, functiesymbolen, relatiesymbolen of combinaties kunnen worden ingevuld. Voor variabelen worden voornamelijk de letters x,y en z gebruikt. Alle symbolen kunnen met een index voorkomen (bijvoorbeeld x_1, a_6 etc.)

DEFINITIE
Een *databaseskelet* is een triple S = <F, P, ar>, waarbij:

F een verzameling functiesymbolen
P een verzameling relatiesymbolen
ar: $F \cup P \rightarrow \mathbb{N}$ de ariteitsfunctie.

De ariteitsfunctie geeft voor ieder functie- en relatiesymbool de ariteit weer.
□

Deze definitie van een databaseskelet uit (Lloyd, 1987) wijkt iets af van die uit (De Brock, 1990), doordat in de hier gepresenteerde definitie domeindefinities niet tot het databaseskelet worden gerekend. In navolging van (Pels, 1988) worden domeindefinities beschouwd als constraints.

VOORBEELD A.0
Een databaseskelet voor de gegevens over de Koninklijke Familie kan er als volgt uitzien:

F = { claus, beatrix, irene, margriet, pieter, verjaardag }
P = { vader, getrouwd, kind }
ar(claus) = 0, ar(beatrix) = 0, ar(irene) = 0, ar(margriet) = 0, ar(pieter) = 0,
ar(verjaardag) = 1, ar(vader) = 2, ar(getrouwd) = 2, ar(kind) = 2.

De functie "verjaardag" geeft voor iedere persoon (een constante) de datum (dag/maand) van de verjaardag van de betreffende persoon.
□

DEFINITIE
Een *term* wordt recursief gedefinieerd als volgt:

- Iedere constante is een term
- Iedere variabele is een term
- Voor ieder functiesymbool f met ar(f) = n en termen $t_1,..,t_n$ geldt: $f(t_1,..,t_n)$ is een term
- Er zijn geen andere termen

Een term zonder een variabele wordt een *grondterm* genoemd.
□

Termen kunnen worden gecombineerd met een relatiesymbool. Er ontstaat dan een *atoom*:

DEFINITIE
Voor een relatiesymbool p met ariteit n (ar(p) = n) en termen $t_1,..,t_n$ geldt: $p(t_1,..,t_n)$ is een atoom.

Als geen van de termen uit een atoom een variabele bevat, wordt de atoom een *grondatoom* genoemd.
□

Atomen kunnen worden gecombineerd met het teken '¬'.

DEFINITIE
Als a een atoom is, dan heet a een *positieve uitspraak*, terwijl ¬a een *negatieve uitspraak* wordt genoemd. Analoog aan grondatomen wordt een uitspraak zonder variabelen een *gronduitspraak* genoemd.
□

Voor een databaseskelet S noteren we met T(S), A(S) en L(S) respectievelijk de verzameling van alle termen van S, alle atomen van S en alle uitspraken van S. Het is uit de definities onmiddellijk in te zien, dat geldt:

$A(S) \subset L(S)$: een atoom is een positieve uitspraak.

DEFINITIE
Een *expressie* (synoniem: formule) wordt recursief gedefinieerd als:

- Iedere uitspraak is een expressie.
- Voor expressies p en q zijn ook $p \wedge q$, $p \vee q$ expressies.
- Voor een expressie p, die een variabele x bevat, zijn $\forall x{:}p$ en $\exists x{:}p$ ook expressies. Dit type expressies wordt ook wel *gekwantificeerde expressie* genoemd.
- Er zijn geen andere expressies

Voor de expressie $\neg p \vee q$ wordt ook wel genoteerd $p \rightarrow q$.

Een expressie zonder variabelen heet een *grondexpressie*.
□

NOTATIE
Voor een expressie L geldt:

~L: de ontkenning van L.

□

DEFINITIE

Stel p een expressie met variabelen $x_1,..,x_n$ (alle verschillend). Stel $c_1,..,c_n$ termen, waarbij iedere c_i verschilt van iedere x_j. Een ***substitutie*** van x_1 door c_1, x_2 door c_2,..,x_n door c_n in p wordt verkregen door alle voorkomens van x_1 te vervangen door c_1, van x_2 door c_2,.., van x_n door c_n. Als alle c_i's grondtermen zijn, wordt de substitutie een ***grondsubstitutie*** genoemd. Een substitutie zoals deze wordt genoteerd als $\{x_1\backslash c_1,..,x_n\backslash c_n\}$. Het toepassen van een substitutie $\vartheta = \{x_1\backslash c_1,..,x_n\backslash c_n\}$ op een expressie p als hierboven wordt genoteerd als $p\vartheta$ of als $p\{x_1\backslash c_1,..,x_n\backslash c_n\}$. Als $v = \{L_1,..,L_n\}$ een verzameling expressies is en ϑ een substitutie, dan is kortweg $v\vartheta = \{L_1\vartheta,..,L_n\vartheta\}$

□

DEFINITIE

Als E een verzameling expressies is en F_0 een verzameling constanten, dan is $[E]_{F0}$ de verzameling van alle grondexpressies, die te verkrijgen zijn door alle mogelijke substituties van variabelen uit expressies door constanten uit F_0.
Als de verzameling constanten verder niet gedefinieerd is, dan wordt genoteerd: [E].

Een bijzonder geval van deze notatie is [L(S)]. Hiermee wordt de verzameling van alle gronduitspraken bij een databaseskelet S weergegeven.

□

VOORBEELD A.1

Stel E = {zoon(x,claus)} en F_0 = {willem-alexander,claus,juliana}
Dan:
$[E]_{F0}$ = { zoon(willem-alexander,claus), zoon(claus,claus), zoon(juliana,claus) }

□

Als de betekenis van een expressie bekend is, kan worden bepaald, of de expressie al dan niet een ware expressie is. Wat 'waar' is in deze context, is afhankelijk van het deel van de wereld, dat door de expressie wordt gemodelleerd. In dit proefschrift wordt niet verder stilgestaan bij de wijze, waarop een betekenis aan een expressie wordt toegekend (zie bijvoorbeeld (Lloyd, 1987)), noch bij de wijze, waarop een waarheidswaarde van een expressie wordt bepaald. Er wordt aangenomen, dat er een betekenistoekennende functie bestaat en dat het voor iedere (grond)expressie mogelijk is een waarheidswaarde te bepalen.

A.2 DEFINITIE VAN EEN KENNISMODEL

In de rest van dit hoofdstuk wordt verondersteld, dat er een databaseskelet S bekend is. In deze paragraaf zal worden uitgewerkt wat een kennissysteem zou moeten kunnen berekenen, gegeven dat databaseskelet. Zoals in hoofdstuk 2 van dit proefschrift al is vermeld, wordt een kennissysteem vaak aangeduid met "feiten en regels". De op te bouwen definitie zal aansluiten bij deze informele weergave van een kennissysteem. Vandaar dat allereerst aandacht zal worden geschonken aan de "feiten". Deze worden verondersteld aanwezig te zijn in een database.

DEFINITIE
Gegeven het databaseskelet S wordt een *databaseuniversum* (DB-universum) behorend bij S als volgt gedefinieerd:

$$U(S) = \{\ u \subseteq [L(S)] \mid \text{er is geen } L \in [L(S)] \text{ zodat } L \in u \text{ en } \sim L \in u\ \}$$

Een element u van U(S) wordt een *databasetoestand* (DB-toestand of toestand) genoemd. Om typografische redenen zal zo'n element worden aangegeven met "u".
□

Er wordt ervan uitgegaan, dat een expressie, die zich in een DB-toestand "u" bevindt, een waarheidswaarde TRUE heeft. Met andere woorden: alleen ware expressies worden in een database opgenomen.

U(S) bestaat uit alle mogelijke deelverzamelingen "u" van de verzameling gronduitspraken bij S zodanig dat ze *consistent* is: als L een element van "u" is, dan is ~L geen element van "u" en omgekeerd. (Met de afspraak uit de voorgaande alinea zou aanwezigheid van zowel L als ~L in "u" betekenen, dat beide grondexpressies waar zijn).

Het toelaten van alleen grondexpressies in "u" is geen ernstige beperking. Het is te allen tijde mogelijk om een niet-gronduitspraak L te vervangen door de elementen uit [L].

In de praktijk zijn er slechts een beperkt aantal toestanden, die op een juiste wijze een deel van de werkelijkheid modelleren. Om aan te geven, aan welke voorwaarden een toestand moet voldoen om in die zin correct te zijn, wordt het begrip *integriteitsconstraint* gebruikt (afgekort tot constraint):

DEFINITIE
Stel gegeven een DB-universum U(S). Een constraint is een functie
c: U(S) → { TRUE, FALSE, UNDEF }

Dit type constraint wordt ook wel een *statische constraint* genoemd.
□

Behalve statische constraints kunnen ook dynamische constraints worden onderscheiden. Een dynamische constraint kan echter worden beschouwd als een statische constraint door een tijdsaanduiding mee te geven. In het vervolg zullen daarom alleen statische constraints worden onderscheiden.

Een constraint c wordt meestal weergegeven dmv. een expressie φ. Als φ een logische consequentie is van de toestand u, dan geldt c(u) = TRUE. Als ~φ een logische consequentie is van de toestand u, dan geldt c(u) = FALSE. In alle andere gevallen geldt: c(u) = UNDEF. Vaak wordt geen onderscheid gemaakt tussen de functie c en zijn bijbehorende expressie: φ wordt dan (slordig) een constraint genoemd.

Merk op, dat de waarde UNDEF (van Undefined) ook in het waardenbereik van c zit. Deze waarde wordt aangenomen, als niet kan worden bepaald of een toestand "u" TRUE of FALSE is voor een zekere constraint c. Dit kan het geval zijn als er niet genoeg informatie aanwezig is, om de waarde van c te bepalen.

VOORBEELD A.2
Stel U(S) ={ u_1, u_2, u_3 }, met

u_1={ vader(claus,willem-alexander), zoon(willem-alexander,claus) }
u_2={ vader(claus,willem-alexander),¬zoon(willem-alexander,claus) }
u_3={ vader(claus,willem-alexander) }

Stel c de constraint, voorgesteld door de expressie φ:
∀x,y:vader(x,y)→zoon(y,x).

Dan geldt:
$c(u_1)$ = TRUE
$c(u_2)$ = FALSE
$c(u_3)$ = UNDEF

In het laatste geval ontbreekt de informatie betreffende het al dan niet voldaan zijn aan c. Vandaar dat geen uitspraak kan worden gedaan over het TRUE of FALSE zijn van de toestand u_3.
□

DEFINITIE
Stel U(S) een DB-universum en C een verzameling constraints. Een $u \in U(S)$ heet een *toegelaten (DB-)toestand* met betrekking tot C, als het volgende geldt:

$$\forall\, c \in C\text{: } c(u) = \text{TRUE} \vee c(u) = \text{UNDEF}.$$

Het *toegelaten DB-universum* met betrekking tot C is:

$$\{\, u \in U(S) \mid u \text{ is een toegelaten DB-toestand met betrekking tot C} \,\}$$

□

Tenzij anders vermeld zullen in het vervolg alleen toegelaten DB-toestanden en toegelaten DB-universa worden verondersteld.

Elementen van een DB-toestand hebben vaak een bepaalde relatie tot elkaar. Zo'n relatie kan in de vorm van een voorschrift ("regel") worden vastgelegd. Als het voorschrift zodanig is, dat uit de waarde van enkele elementen uit de DB-toestand de waarde van andere elementen af te leiden is, kunnen de afleidbare elementen uit de DB-toestand worden verwijderd en kan in plaats daarvan het algemene voorschrift worden gebruikt om op ieder gewenst moment de afleidbare waarden te berekenen.

In het vervolg wordt verondersteld, dat zo'n voorschrift te geven is met behulp van een regel:

DEFINITIE
Een *regel* is een expressie van de vorm

$$\forall x_1,..,\forall x_s : L_1 \vee .. \vee L_n$$

met alle L_i uitspraken, alle x_i variabelen, elk voorkomend in $L_1 \vee .. \vee L_n$. Verder wordt verondersteld, dat $s \geq 1$: een regel bevat minstens 1 variabele.
□

Zonder verlies van algemeenheid mag worden verondersteld dat iedere regel als volgt te noteren is:

$$\forall x_1,..,\forall x_s : \neg A_1 \vee ... \vee \neg A_m \vee B_1 \vee ... \vee B_n$$

(Met eventueel m=0 of n=0).

Een regel van deze vorm zal voortaan worden genoteerd als:

$$A_1 \wedge ... \wedge A_m \rightarrow B_1 \vee ... \vee B_n \text{ of } A_1,..,A_m \rightarrow B_1,..,B_n$$

Hierbij is iedere A_i en B_j een atoom. Het geval n=0, wordt verderop behandeld. Als m=0 dan heet deze regel een *feit.* In dat geval wordt de '$\rightarrow$' weggelaten. Als m=0 en n=0, dan heet deze regel een *lege regel.* $A_1 \wedge ... \wedge A_m$ heet de *staart* van de regel, $B_1 \vee ... \vee B_n$ heet de *kop* van de regel. Merk op, dat in de kop van de regel alleen positieve uitspraken staan.

De begrippen atoom en feit zijn synoniem. Vaak wordt het begrip feit gebruikt in een databasecontext.

Vaak (bijvoorbeeld in (Lloyd, 1987)) wordt slechts een beperkte set regels beschouwd, namelijk die regels met n≤1. Zo'n regel wordt dan wel een *Hornclause* genoemd. Het voordeel van deze beperking is, dat het inferentieproces met behulp van de resolutieregel en de 'negation as finite failure rule' bewijsbaar juist en volledig is. Bijvoorbeeld de taal Prolog is hierop gebaseerd.

DEFINITIE
Stel gegeven een DB-toestand "u" en een regel r:
$A_1 \wedge ... \wedge A_m \rightarrow B_1 \vee ... \vee B_n$

Het *toepassen van r op u,* notatie r(u), wordt gedefinieerd als volgt:

$$r(u) = u \cup \{(B_1 \vee ... \vee B_n)\vartheta \mid \vartheta \text{ is een substitutie, } \{A_1\vartheta,...,A_m\vartheta\} \subseteq u \}$$

□

Omdat verondersteld werd, dat een DB-toestand alleen gronduitspraken zou bevatten, mag ervan worden uitgegaan, dat de substitutie ϑ uit de definitie een grondsubstitutie is. Omdat verder in de DB-toestand alleen uitspraken worden toegestaan (zie de eerdere definitie van

DB-toestand), geldt de definitie van r(u) alleen voor regels r, die slechts één uitspraak in de kop hebben staan.

Uitbreidingen van databases, die ontstaan door ook expressies van de vorm $B_1 \vee ... \vee B_n$ in de kop van een regel toe te staan, zijn diepgaand bestudeerd in de literatuur (zie bijvoorbeeld (Levesque, 1984)). Dit wordt wel het probleem van *incomplete informatie* genoemd: er is immers alleen bekend, dat een of meer van de B_i's mag worden geconcludeerd, maar niet welke. Levesque (1984) heeft aangetoond, dat, om een database met incomplete informatie te specificeren en te bevragen, een taal nodig is, die zowel naar de kennis over de database als naar de toestand van de database kan refereren. Ander gezegd: een systeem moet kennis hebben over zijn eigen kennis om vragen aangaande incomplete informatie correct te beantwoorden. Het volgende voorbeeld (ontleend aan (Levesque, 1989)) verduidelijkt dit:

VOORBEELD A.3
Bij dit voorbeeld wordt ervan uitgegaan, dat een DB-toestand ook expressies, anders dan uitspraken, mag bevatten.

Stel de DB-toestand u_1:
u_1 = { vader(claus,johan-friso), vader(claus,willem-alexander) $\vee$ vader(claus,jaime) }

Bekijk de volgende vragen en antwoorden:

Van wie is claus de vader?:
vader(claus,y). Antwoord: y = johan-friso.
Dit is geen complete lijst (maar de gebruiker weet dat niet en kan daar ook niet door een vraag achter komen).

Is er iemand, van wie claus de vader is, anders dan van johan-friso?
$\exists$y:vader(claus,y) $\wedge$ y $\neq$ johan-friso. Antwoord: ja.
Maar het systeem weet de naam van die persoon niet.

Stel, dat de vraagtaal wordt uitgebreid met een operator K met de (informele) betekenis:

K[α] : het systeem weet dat α geldt

Bekijk nu de volgende vragen en antwoorden:

Van wie is claus de vader?:
vader(claus,y). Antwoord: y = johan-friso.
Maar:
Ken je alle personen, van wie claus de vader is?
K[$\forall$y:vader(claus,y)]. Antwoord: nee. (Door deze vraag kan de gebruiker er dus achter komen, dat de vorige vraag een niet-compleet antwoord had.)

Is er iemand, van wie claus de vader is, anders dan van johan-friso?
$\exists$y:vader(claus,y) $\wedge$ y $\neq$ johan-friso. Antwoord: ja.
Maar:
Ken je een persoon, anders dan johan-friso, van wie claus de vader is?
$\exists$y:K[vader(claus,y)] $\wedge$ y $\neq$ johan-friso. Antwoord: nee.
En:
Is er een persoon, van wie claus de vader is en die je niet kent?
$\exists$y:vader(claus,y) $\wedge$ $\neg$K[vader(claus,y). Antwoord: ja. (De laatste twee vragen geven de gebruiker de informatie, dat het systeem de namen van de personen, waarvan claus de vader is, niet kent en dat het systeem dat ook weet.)

Het toevoegen van de operator K geeft dus de mogelijkheid om het systeem te vragen naar zijn kennis over zijn eigen kennis. Hierdoor wordt een meer compleet beeld verkregen van de kennis in de database.
□

Omdat het toevoegen van incomplete informatie aan databases een studie op zich is, die te diepgaand is voor het onderwerp van deze studie zullen in het vervolg alleen Hornclauses worden beschouwd. Een speciaal geval van deze beperkingen betreffen die regels, waarbij een negatieve uitspraak in de kop van de regel staat. Als p een relatiesymbool is en $\{a_1,..,a_k\}$ de verzameling constanten, die p als argument kan hebben en er geldt tenminste een van de uitspraken $p(a_1),...,p(a_k)$, dan is de regel $c_1,..,c_m \rightarrow \neg p(a_1)$ equivalent met de regel
$c_1,..,c_m \rightarrow p(a_2) \vee ... \vee p(a_k)$.

Veel tools voor de implementatie van kennissystemen staan ook regels toe van de vorm:

$$A_1 \wedge ... \wedge A_m \rightarrow B_1 \wedge ... \wedge B_n$$

Deze regel is echter equivalent met de volgende verzameling Hornclauses:

$$\{ \quad A_1 \wedge ... \wedge A_m \rightarrow B_1, ..., A_1 \wedge ... \wedge A_m \rightarrow B_n \}$$

Het is vanuit dat oogpunt bezien dan ook geen ernstige beperking om alleen Hornclauses te beschouwen.

Een verzameling regels zal voortaan een *regelbank* of *rulebase* worden genoemd (notatie: R). Er zal steeds worden aangenomen, dat de predicaatsymbolen en de functiesymbolen, die in de regels worden gebruikt, aanwezig zijn in het databaseskelet. Anders gezegd: symbolen, die in regels worden gebruikt zijn bekende symbolen.

Een *kennisbank* wordt in de literatuur vaak beschreven als 'feiten en regels'. Dit begrip blijkt ook uit de nu volgende definitie:

DEFINITIE
Gegeven een toegelaten DB-universum U en een regelbank R. Een kennisbank is een tupel <u,R>, waarbij u $\in$ U een (toegelaten) DB-toestand is.
□

Stel gegeven een kennisbank <u,R>. De regels uit R kunnen worden toegepast op de DB-toestand "u" om nieuwe feiten af te leiden en, door de nieuwe feiten aan de DB-toestand "u" toe te voegen, een nieuwe DB-toestand u' te krijgen. (Eigenlijk ontstaat zo een nieuwe kennisbank <u',R>. Omdat de regelbank R bij dit proces echter geen veranderingen ondergaat, zal steeds alleen de DB-toestand worden beschouwd). Ook u' moet een toegelaten DB-toestand zijn. Dit proces van "toepassen van regels op een DB-toestand" kan weer worden herhaald op u' om zo u" te krijgen etc. Het proces stopt als er geen nieuwe feiten meer kunnen worden afgeleid. Dit kan als volgt worden gedefinieerd:

DEFINITIE
Stel gegeven een kennisbank <u,R>. De *deductieve afsluiting* van "u" onder R is een verzameling u ∪ v, zodat:

- u ∪ v ⊆ [L(S)] (u ∪ v bevat alleen uitspraken)
- u ∪ v ∈ U (u ∪ v moet een toegelaten DB-toestand zijn).
- $v_1 \in v \Leftrightarrow \exists r_1,..,r_j \in R: v_1 \in r_j(..(r_1(u)))$. (Elementen uit v zijn alleen te krijgen door herhaald toepassen van regels uit R op de uitgangstoestand u)

In de derde eis garandeert de "⇐" dat ieder feit dat door het toepassen van een of meerdere regels uit R uit "u" kan worden afgeleid, element is van v. Dit wordt ook wel aangeduid met "u ∪ v is maximaal binnen [L(S)]"
De verzameling v zal voortaan de uit "u" onder R *afleidbare* verzameling feiten worden genoemd. In (Lloyd, 1987) wordt dit begrip de transitieve afsluiting genoemd.
□

Voorlopig wordt ervan uitgegaan, dat alle feiten, die afgeleid worden bij de berekening van de deductieve afsluiting, niet in strijd zijn met reeds bestaande feiten uit "u" of eerder afgeleide feiten. Dit rechtvaardigt het gebruik van "⇔" in de derde voorwaarde van de definitie. In het vervolg wordt aangenomen, dat het systeem de beschikking heeft over een mechanisme om te controleren of een DB-toestand aan alle constraints voldoet.

DEFINITIE
Stel gegeven een DB-universum U en een verzameling constraints C. Een *controle mechanisme* is een functie

$$CM: U \rightarrow \{ TRUE, FALSE \}$$

gedefinieerd door:

$CM(u) = TRUE$ als $\forall c \in C: c(u) = TRUE \vee c(u) = UNDEF$
$CM(u) = FALSE$ anders

□

Over de implementatie van de functie CM worden geen uitspraken gedaan.

Het begrip "deductieve afsluiting" zou waardeloos zijn, als bij een gegeven regelbank en DB-toestand meerdere deductieve afsluitingen mogelijk zouden zijn. Vandaar de volgende stelling:

STELLING A.1
Gegeven een DB-universum U en een $u \in U$. Verder zij gegeven een regelbank R. Dan is de deductieve afsluiting van "u" onder R uniek.

Bewijs
Stel v' en v" twee uit "u" onder R afleidbare verzamelingen. Stel $p \in v'$. Per definitie bestaat er een rij regels $r_1,..,r_j$ uit R zodanig dat $p \in r_j(..(r_1(u)))$. Per definitie geldt dan ook: $p \in v"$. Dus $v' \subseteq v"$. Analoog geldt: $v" \subseteq v'$. Dus $v'=v"$. QED
□

Gegeven een kennisbank (u,R) is het dus geoorloofd om te spreken van *de* deductieve afsluiting van "u" onder R. Dit kan worden weergegeven met behulp van een functie:

DEFINITIE
Een *kennisfunctie* k_R is een functie

$$k_R: U \rightarrow U$$

gedefinieerd door: $k_R(u)$ = de deductieve afsluiting van "u" onder R.
□

De volgende stellingen geven enkele eigenschappen van de functie k_R.

STELLING A.2
Stel $u_1 \subseteq u_2$ twee DB-toestanden. Stel R een regelbank. Dan geldt: $k_R(u_1) \subseteq k_R(u_2)$.

Bewijs
Stel $p \in k_R(u_1)$. Per definitie zijn er dan $r_1,..,r_j$ uit R zodat $p \in r_j(..(r_1(u_1)))$. maar dan ook (vanwege $u_1 \subseteq u_2$) $p \in r_j(..(r_1(u_2)))$. Dus $p \in k_R(u_2)$. QED.
□

COROLLARIUM A.1
Als $u_1 \subseteq k_R(u_2)$, dan ook $k_R(u_1) \subseteq k_R(u_2)$. (Neem in stelling A.2 $u_2 = k_R(u_2)$).
□

COROLLARIUM A.2

Stel $u_1 \subseteq u_2 \subseteq k_R(u_1)$. Dan $k_R(u_1) = k_R(u_2)$. (Stelling A.2 geeft $k_R(u_1) \subseteq k_R(u_2)$. Corollarium A.1 geeft $k_R(u_2) \subseteq k_R(u_1)$).

□

Het volgende voorbeeld toont aan, dat de eis $u_2 \subseteq k_R(u_1)$ bij corollarium A.2 geen overbodige eis is.

VOORBEELD A.4

Stel u_1 = { zoon(johan-friso,claus) }

u_2 = { zoon(johan-friso,claus), dochter(beatrix,juliana),vrouw(juliana) }

R = { dochter(x,y) $\wedge$ vrouw(y) $\rightarrow$ moeder(y,x) }

Dan $u_1 \subseteq u_2$, $k_R(u_1) = u_1$, $k_R(u_2) = u_2 \cup$ {moeder(juliana,beatrix)}, $u_2 \not\subseteq k_R(u_1)$ en $k_R(u_1) \neq k_R(u_2)$.

□

Op dit moment zijn er nog geen uitspraken gedaan, HOE de consistentie en toelaatbaarheid wordt gecontroleerd, noch hoe de deductieve afsluiting wordt berekend. Er wordt slechts aangenomen, dat deductieve afsluitingen kunnen worden berekend.

Gegeven een kennisbank <u,R> kan de rol van een regelbank R dus worden omschreven als: het berekenen van de deductieve afsluiting van "u". In feite is de deductieve afsluiting van "u" onder R de betekenis van de kennisbank <u,R>. Dit komt tot uitdrukking in de volgende definitie van een kennismodel:

DEFINITIE

Een *kennismodel* is een tupel <S,C,U,R,k_R>, waarbij

- S een databaseskelet
- C een verzameling constraints
- U = U(S) het toegelaten DB-universum (m.b.t. C)
- R een regelbank
- $k_R : U \rightarrow U$ de kennisfunctie

□

Een kennismodel geeft als het ware de "buitenkant" van een kennissysteem weer. Er wordt aangegeven WAT met behulp van een regelbank wordt berekend, niet HOE het systeem deze berekening uit kan voeren.

DEFINITIE

Gegeven een kennismodel $<S,C,U,R,k_R>$. Stel $<u,R>$ een kennisbank voor een $u \in U$. Van de verzameling [L(S)] van alle gronduitspraken kan dan de volgende partitie worden gemaakt:

- $k_R(u)$ is de maximale verzameling uitspraken, die vanuit "u" door R kan worden berekend. De verzameling "u" bevat de expliciet opgeslagen feiten, $k_R(u) \backslash u$ is de uit "u" onder R afleidbare verzameling. Alleen uitspraken uit $k_R(u)$ hebben een waarheidswaarde TRUE.
- $\{ \sim L \mid L \in k_R(u) \}$ is de verzameling *verboden uitspraken*. Vanwege de consistentie van $k_R(u)$ hebben alle uitspraken uit $\{ \sim L \mid L \in k_R(u) \}$ de waarheidswaarde FALSE.
- $k_R(u) \cup \{ \sim L \mid L \in k_R(u) \}$ is het *bereik van kennis* van de kennisbank $<u,R>$. Deze verzameling bevat alle uitspraken, waarvan de waarheidswaarde kan worden bepaald.
- $L(S) \setminus (k_R(u) \cup \{ \sim L \mid L \in k_R(u) \})$ is de verzameling *ontoegankelijke uitspraken*. Over de uitspraken uit deze verzameling kan geen uitspraak worden gedaan, gebaseerd op de kennisbank $<u,R>$ en de kennisfunctie k_R. Deze verzameling is gelijk aan de verzameling $\{ L \in [L(S)] \mid L \notin k_R(u) \wedge \sim L \notin k_R(u) \}$. Deze verzameling wordt voortaan genoteerd als O(u,R).

In figuur A.1 is dit verduidelijkt.

Het volgende voorbeeld geeft een illustratie van de hiervoor gedefinieerde begrippen.

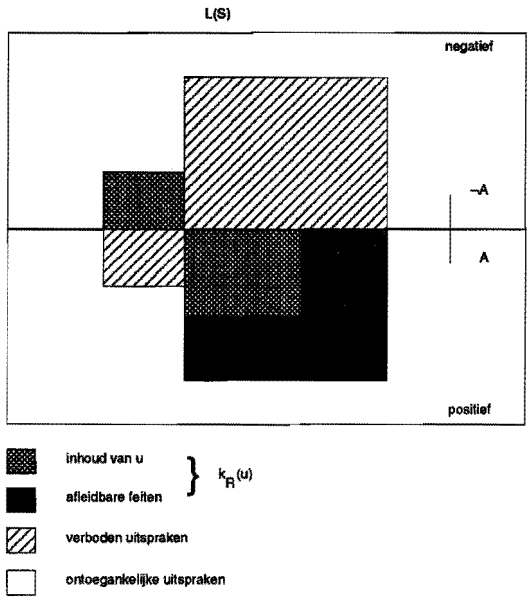
VOORBEELD A.5

Stel gegeven de kennisbank $<u,R>$ met de volgende inhoud:

$u = \{$ vader(bernhard,beatrix), $\neg$moeder(juliana,jaime), vrouw(beatrix) $\}$
$R = \{$ vader(x,y)$\wedge$vrouw(y) $\rightarrow$ dochter(y,x) $\}$

Stel verder de verzameling constraints $C=\varnothing$. Dan geldt:

De deductieve afsluiting $k_R(u) = u \cup \{$ dochter(beatrix,bernhard) $\}$



Figuur A.1 Partitie van L(S)

De verzameling verboden uitspraken is:

{ ¬vader(bernhard,beatrix), moeder(juliana,jaime), ¬vrouw(beatrix), ¬dochter(beatrix,bernhard) }

Het bereik van kennis is:

{ vader(bernhard,beatrix),¬moeder(juliana,jaime), vrouw(beatrix), dochter(beatrix,bernhard), ¬vader(bernhard,beatrix), moeder(juliana,jaime), ¬vrouw(beatrix), ¬dochter(beatrix,bernhard) }

Alle overige uitspraken zijn de ontoegankelijke uitspraken. Gegeven de eerdere predicaatsymbolen en constanten is bijvoorbeeld "vrouw(juliana)" een ontoegankelijke uitspraak: uit de inhoud van de DB-toestand "u" en de regelbank R kan de waarheidswaarde van deze expressie niet worden afgeleid.

□

Een kennisbank <u,R> is dus in feite een representatie van zijn deductieve afsluiting. Een kennissysteem moet in staat zijn deze deductieve afsluiting te berekenen. Een kennisbank kan daarom als het statische deel van een kennissysteem worden beschouwd. In de volgende paragraaf zal de dynamiek van een kennissysteem worden gedefinieerd.

A.3 DYNAMIEK VAN EEN KENNISSYSTEEM

In deze paragraaf zullen de componenten worden gedefinieerd, waarmee een kennissysteem in staat is de deductieve afsluiting van een gegeven kennisbank <u,R> te berekenen. Er zal steeds van worden uitgegaan dat een kennismodel $<S,C,U,R,k_R>$ gegeven is. Allereerst volgt een informele definitie van het begrip *query*.

DEFINITIE
Een query is een verzameling van uitspraken.
□

Informeel kan de werking van een kennissysteem als volgt worden beschreven. Aan een kennisbank wordt een query aangeboden. Het systeem heeft de taak om van iedere gronduitspraak uit de query de waarheidswaarde te bepalen. Dit gebeurt door na te gaan, in welke van de deelverzamelingen uit paragraaf A.2 een gronduitspraak aanwezig is. Voor een niet-gronduitspraak bepaalt het systeem alle mogelijke substituties zodanig, dat een gronduitspraak ontstaat, die in de deductieve afsluiting van de kennisbank zit (en die dus de waarheidswaarde TRUE heeft). Het volgende voorbeeld is een illustratie van deze informeel geformuleerde werking van een kennissysteem.

VOORBEELD A.6
Stel gegeven de kennisbank <u,R> met de volgende inhoud:

u = { vader(jorge,bernardo), vader(jorge,juliana-jr), vader(jorge,nicolas), vader(pieter,floris), man(floris), man(nicolas), getrouwd(jorge,christina) }
R = { vader(x,y)∧man(y) → zoon(y,x) }

Stel gegeven de query: { zoon(x,y)∧getrouwd(y,christina) } (Geef de namen van de zonen van diegene, die getrouwd is met christina).

Berekening van $k_R(u)$ levert op:

$k_R(u)$= { vader(jorge,bernardo), vader(jorge,juliana-jr), vader(jorge,nicolas), vader(pieter,floris), man(floris), man(nicolas), getrouwd(jorge,christina), zoon(floris,pieter), zoon(nicolas,jorge) }

De enig mogelijke substitutie van de uitspraken in de query, die uitspraken uit $k_R(u)$ opleveren, is: {x\nicolas, y\jorge}.

Merk op, dat er meer wordt verkregen, dan informeel is geformuleerd: de naam van de man van christina is er 'gratis' bijgeleverd. Dit komt door de vertaling van de informeel gestelde query naar de formeel gestelde query. Bij die vertaling is weggevallen, dat er alleen interesse bestond voor de namen van de zonen.

Beschouw vervolgens de query: {zoon(juliana-jr,jorge)}. Omdat deze gronduitspraak in de verzameling ontoegankelijke uitspraken zit (zoals makkelijk is na te gaan), kan het systeem deze query niet beantwoorden: de waarheidswaarde van deze uitspraak is onbepaald.
□

Een probleem apart vormt de aanwezigheid van negatieve uitspraken in de query. Omdat er geen negatieve uitspraken kunnen worden afgeleid, kan slechts in twee gevallen de waarheidswaarde van een negatieve uitspraak ¬L in een query op een kennisbank <u,R> worden bepaald:

1. ¬L komt al voor in de DB-toestand "u". ¬L heeft dan waarheidswaarde TRUE.
2. L komt voor in $k_R(u)$. ¬L heeft dan waarheidswaarde FALSE.

In alle andere gevallen is de waarheidswaarde voor een negatieve uitspraak niet te bepalen. Informeel gesteld zal het kennissysteem een query met een negatieve uitspraak ¬L oplossen door (als ¬L niet voorkomt in u), naar de aanwezigheid van L te zoeken in $k_R(u)$.

VOORBEELD A.7
Stel de kennisbank <u,R> als in het vorige voorbeeld. Beschouw de query
{ zoon(x,y)∧ ¬getrouwd(y,christina)}.

Omdat in $k_R(u)$ het feit 'getrouwd(jorge,christina)' aanwezig is, leidt het systeem af, dat '¬getrouwd(jorge,christina)' de waarheidswaarde FALSE heeft. Omdat er verder geen uitspraken betreffende 'getrouwd' zijn, levert het systeem de lege verzameling als antwoord: er zijn geen substituties voor x en y te vinden. Merk op, dat het systeem niet in staat is met de inhoud van deze kennisbank af te leiden, dat het substitueren van y=pieter een goede

oplossing kan leveren. Het 'weet' immers niet, dat in het Koninklijke Huis iedereen met maximaal een persoon is getrouwd.

Beschouw nu de query {¬zoon(x,pieter)}. Omdat in $k_R(u)$ het feit zoon(floris,pieter) aanwezig is, zal het systeem niet als oplossing {x\floris} geven. Het is echter niet in staat om met de inhoud van deze kennisbank af te leiden, dat een oplossing kan zijn {x\nicolas}, hoewel zoon(nicolas,jorge) als feit in $k_R(u)$ voorkomt. Het 'weet' immers niet, dat ieder persoon slechts 1 vader heeft.
□

Na deze voorbeelden wordt nu de dynamiek van een kennissysteem formeel gedefinieerd. Allereerst de formele definitie van het eerder informeel gedefinieerde begrip "query".

DEFINITIE
Een query (synoniem: goal) is een conjunctie $g = L_1 \wedge .. \wedge L_n$. $L_i \in L(S)$. Een uitspraak in een query wordt een *hypothese* genoemd.

Een query $g = L_1 \wedge .. \wedge L_n$ zal ook worden genoteerd als $g = \{L_1,..,L_n\}$.
□

In het vervolg zal ook het begrip "onafhankelijkheid van hypothesen" nodig zijn.

DEFINITIE
Stel $g = \{L_1,..,L_n\}$ een query. Stel $\{x_{11},..,x_{1t1}\}$ de variabelen binnen L_1, $\{x_{21},..,x_{2t2}\}$ de variabelen binnen L_2,.., $\{x_{n1},..,x_{ntn}\}$ de variabelen binnen L_n. Dan:

$L_1,..,L_n$ *onafhankelijk* $\Leftrightarrow \forall i \forall j: i \neq j \Rightarrow \{x_{i1},..,x_{iti}\} \cap \{x_{j1},..,x_{jtj}\} = \varnothing$.

Als een verzameling hypothesen niet onafhankelijk is, dan is ze *afhankelijk.*
□

Merk op, dat iedere grondhypothese onafhankelijk is van iedere andere hypothese. Tevens kunnen binnen een verzameling afhankelijke hypothesen deelverzamelingen van onafhankelijke hypothesen voorkomen.

Het is voor diverse in het navolgende te bewijzen stellingen handig, indien een query gesplitst kan worden in een aantal deelqueries, zodanig, dat twee hypothesen, die elk uit verschillende deelqueries komen, onafhankelijk zijn. Dit gebeurt door de volgende definitie.

DEFINITIE

Stel $g = \{L_1,..,L_n\}$ een query. Dan $\{g_1,...g_k\}$ is een *orthogonale splitsing* van g als:

a. $g_i \cap g_j = \emptyset$, $i \neq j$
b. $g = \bigcup_i g_i$
c. $g_i \neq \emptyset$, alle i.
d. $\forall s \forall t$: $s \neq t \Rightarrow \forall L_i \in g_s \; \forall L_j \in g_t$: L_i en L_j onafhankelijk
e. $\forall s$: $|g_s| \geq 2 \Rightarrow \forall L_i \in g_s \; \exists L_j \in g_s$: $L_i \neq L_j$ en L_i en L_j afhankelijk

Een element uit een orthogonale splitsing van g is een *deelquery* van g.
□

Uit de eisen a, b en c volgt direct, dat een orthogonale splitsing van g een partitie van g vormt. Deze partitie wordt bepaald door deelqueries te vormen zodanig dat er tussen de deelqueries geen gemeenschappelijke variabelen voorkomen. Bij een gegeven query kunnen meerdere orthogonale splitsingen voorkomen, zoals het volgende voorbeeld aantoont.

VOORBEELD A.8

Stel gegeven de query g = {man(x), vader(x,y), vrouw(z), moeder(z,t)}. Dan zijn de volgende twee verzamelingen os_1 en os_2 orthogonale splitsingen van g.

$os_1 = g$
$os_2 = \{ \{man(x), vader(x,y)\}, \{vrouw(z), moeder(z,t)\}\}$
□

Grondhypothesen vormen altijd aparte deelqueries binnen een orthogonale splitsing, zoals de volgende stelling aantoont.

STELLING A.3

Stel $g = \{L_1,..,L_n\}$ een query en $os = \{g_1,..,g_k\}$ een orthogonale splitsing van g. Stel L_i een grondhypothese. Dan: $g_j = \{L_i\}$ voor een j.

Bewijs

Deze stelling volgt direct uit eis e. van de definitie van orthogonale splitsing en de observatie, dat iedere gronduitspraak onafhankelijk is van iedere andere uitspraak.
□

Door het introduceren van het begrip orthogonale splitsing van g kan worden vermeden, in het vervolg queries te moeten onderscheiden, waarin zowel grondhypothesen als niet-

grondhypothesen aanwezig zijn. In een orthogonale splitsing worden dat tenminste twee gescheiden "deelqueries", die afzonderlijk kunnen worden opgelost, waarna het antwoord op de query kan worden samengesteld uit die van de deelqueries.

Voordat een antwoord op een query kan worden gedefinieerd, is nog het volgende begrip nodig:

DEFINITIE
Stel Θ en Θ' twee verzamelingen van substituties. Stel verder dat het volgende geldt:
$\forall \vartheta \in \Theta\ \forall \vartheta' \in \Theta'$: $\nexists$x: x\c $\in \vartheta \wedge$ x\c' $\in \vartheta'$. Dan:

$$\Theta \otimes \Theta' = \{\ \vartheta \cup \vartheta' \mid \vartheta \in \Theta, \vartheta' \in \Theta'\ \}$$

Ofwel: door de operator $\otimes$ worden twee substituties "aan elkaar geplakt".
□

VOORBEELD A.9
Stel Θ = {{ x\beatrix, y\claus}} en stel Θ' = {{ z\bernhard }, {z\juliana}}. Dan:
$\Theta \otimes \Theta'$ = { {x\beatrix, y\claus, z\bernhard}, {x\beatrix, y\claus, z\juliana} }
□

Dan volgt nu een formalisatie van eerder gegeven informele beschrijving van de werkwijze van een kennissysteem. Hierbij zal worden begonnen met het begrip "antwoord op een query".

DEFINITIE
Stel $g=\{L_1,..,L_n\}$ een query. Stel os = $\{g_1,..,g_m\}$ een orthogonale splitsing van g. Een *antwoord* op een query is een functie a, die als volgt wordt gedefinieerd:

A. Stel m=1 (g is zijn eigen orthogonale splitsing). Onderscheid:
1. Alle L_i's zijn gronduitspraken
   Dan a:$\wp$L(S)$\rightarrow${TRUE, FALSE, UNKNOWN}. Meer specifiek:
   - $(\forall L_i \in g: L_i \in k_R(u)) \Rightarrow a(\{L_1,..,L_n\})$ = TRUE.
   - $(\exists L_i \in g: \sim L_i \in k_R(u)) \Rightarrow a(\{L_1,..,L_n\})$ = FALSE.
   - $((\{L_i \in g \mid \sim L_i \in k_R(u)\} = \varnothing) \wedge (\exists L_i: L_i \notin k_R(u))) \Rightarrow a(\{L_1,..,L_n\})$ = UNKNOWN.

   (Respectievelijk: als alle L_i uit g binnen $k_R(u)$ zitten, dan is het antwoord TRUE. Als er een L_i uit g element is van de verzameling verboden uitspraken, dan is het antwoord FALSE. Als geen van de L_i's uit g element zijn van de verzameling

verboden uitspraken (dus het antwoord is niet FALSE) en er is een L_i uit g, die element is van de verzameling ontoegankelijke uitspraken, dan is het antwoord UNKNOWN).

2. Veronderstel, dat binnen de query de variabelen $x_1,..,x_s$ voorkomen.
   Dan: $a(g) = \Theta$, waarbij Θ is gedefinieerd als:
   $\Theta = \{ \vartheta \mid \vartheta=\{x_1\backslash c_1,..,x_s\backslash c_s\}$ een grondsubstitutie, $\{L_1\vartheta,..,L_n\vartheta\} \subseteq k_R(u) \}$.

   (Ofwel: het antwoord op de query bestaat uit alle mogelijke substituties binnen de hypothesen, zodat de query een deelverzameling van $k_R(u)$ wordt).

B. Stel m = 2. Stel $a(g_j)$ het antwoord voor g_j (j=1,2).
Dan is het antwoord a(g) uit de volgende tabel af te lezen:

$a(g_2)\downarrow$ $a(g_1)\rightarrow$	TRUE (A)	FALSE (B)	UNKNOWN (C)	Θ (D)
TRUE (1)	TRUE	FALSE	UNKNOWN	Θ
FALSE (2)	FALSE	FALSE	FALSE	$\varnothing$
UNKNOWN (3)	UNKNOWN	FALSE	UNKNOWN	$\varnothing$
Θ' (4)	Θ'	$\varnothing$	$\varnothing$	$\Theta \otimes \Theta'$

(De letters boven iedere kolom en de cijfers naast iedere rij zullen worden gebruikt om naar cellen in de tabel te kunnen verwijzen.)

C. Stel m > 2. Dan kan a(g) worden bepaald door het herhaald gebruiken van de bovenstaande tabel.

In het vervolg zal deze antwoordfunctie de *theoretische antwoordfunctie* worden genoemd.
□

Het is eenvoudig in te zien (door het nagaan van alle gevallen), dat de in de definitie gebruikte tabel "welgedefinieerd" is in het geval, dat g_1 of g_2 alleen grondhypothesen bevatten. Stel dat g_1 alleen grondhypothesen bevat. Stel $\sim L_i \in k_R(u)$ voor een grondhypothese $L_i \in g_1$. Dan $a(g_1)$ = FALSE (kolom (B)). Omdat $g_1 \subseteq g$, geldt $\sim L_i \in k_R(u)$ ook voor een $L_i \in g$. Dus a(g) = FALSE als g_2 ook alleen grondhypothesen bevat (rijen (1), (2) en (3)) en

$a(g) = \varnothing$ als g_2 niet-grondhypothesen bevat: voor geen enkele grondsubstitutie ϑ geldt dan immers: $L_j\vartheta \in k_R(u)$, alle $L_j \in g$ (rij (4)). Analoog voor het geval $a(g_1)$ = TRUE (kolom (A)) en $a(g_2) = \Theta'$ (kolom (D)) (voor alle $L_i \in g_1$ en alle grondsubstituties ϑ geldt immers: $L_i\vartheta \in k_R(u)$, dus $a(g) = \Theta'$).

Stel $a(g_1)$ = UNKNOWN (kolom (C)). Dan is er een $L \in O(u,R)$ met $L \notin k_R(u)$. Stel $a(g_2) = \Theta'$(rij (4)). Dan geldt voor geen enkele grondsubstitutie ϑ dat $L\vartheta \in k_R(u)$. Dit levert daarom het antwoord $a(g) = \varnothing$.

Omdat g_1 en g_2 onafhankelijk zijn, bevatten ze geen gemeenschappelijke variabelen. Het resultaat $\Theta \otimes \Theta'$ voor $a_1(g_1) = \Theta$ en $a_2(g_2) = \Theta'$ is daarom goed-gedefinieerd.

Om een antwoord op een query te verkrijgen, is het vaak nodig, dat het systeem (een deel van) $k_R(u)$ berekent. Dit gebeurt door middel van een redeneerproces. Bij iedere stap uit dat redeneerproces wordt of de DB-toestand, of de verzameling hypothesen gewijzigd. Het is voor de beschrijving van het redeneerproces daarom voldoende om de inhoud van de DB-toestand en de query te beschrijven. Formeel:

DEFINITIE
Een *redeneertoestand* is een paar <u,g>, waarbij:

- $u \in U$ een DB-toestand
- g een query

□

De query g kan op ieder moment worden beschouwd als de verzameling van die hypothesen, waarvoor de waarheidswaarde nog niet is bepaald, cq. waarvoor nog geen substituties zijn gevonden. Anders geformuleerd: in een query zitten die hypothesen, die het berekenen van het antwoord (de bepaling van de functiewaarde voor de theoretische antwoordfunctie a) nog niet mogelijk maken. Het redeneerproces zal daarom pogen de verzameling g leeg te maken. Een redeneerproces bestaat uit een aantal stappen. Bij iedere stap vindt een transformatie plaats van een redeneertoestand naar een andere redeneertoestand. Op deze manier wordt een keten van redeneertoestanden opgebouwd. Formeel:

DEFINITIE

Een *redeneerketen* is een rij $<(u_0,g_0),..,(u_k,g_k)>$, zodat:

(u_i,g_i) is een redeneertoestand (alle i).

Voor alle i, $1 \leq i \leq k$ geldt:

a. $u_{i-1} \subseteq u_i \subseteq k_R(u_{i-1})$

Een van de volgende drie gevallen geldt:

b_1. $g_i = g_{i-1}$

b_2. $g_i = (g_{i-1} \backslash L)\vartheta \wedge (L\vartheta \in k_R(u_0) \vee \sim L\vartheta \in u_0) \wedge L \in g_{i-1}$

b_3. $g_i = (g_{i-1} \cup \{L_1,..,L_n\})\vartheta \wedge L_1,..,L_n \rightarrow A \in R \wedge A\vartheta \in g_{i-1}$

c. $(u_{i-1},g_{i-1}) \neq (u_i,g_i)$

(Met ϑ een grondsubstitutie).

Voor een redeneerketen $<(u_0,g_0),..,(u_k,g_k)>$ zal de overgang van (u_{i-1},g_{i-1}) naar (u_i,g_i) een *redeneerstap* worden genoemd $(1 \leq i \leq k)$.

□

Eis a. geeft aan, dat tijdens een redeneerstap geen informatie verloren gaat. Eisen b_1 t/m b_3 geven aan, wat er met de verzameling hypothesen kan gebeuren. Deze kan onveranderd blijven (b_1); een beantwoorde hypothese kan worden verwijderd (b_2) of er kunnen hypothesen worden toegevoegd (b_3). Dit laatste kan gebeuren als de beantwoording van een hypothese A kan worden afgeleid uit de resultaten van de beantwoording van een aantal andere hypothesen. Het afleiden van het antwoord op hypothese A gebeurt dan door middel van een regel uit R. Eis c. tenslotte is een voortgangseis: er moet iets aan de redeneertoestand veranderd zijn tijdens de redeneerstap.

Merk op, dat eis a. door het herhaald toepassen van corollarium A.2 gelijkwaardig is met:

a. $u_{i-1} \subseteq u_i \subseteq k_R(u_0)$

Ofwel: door het opbouwen van een redeneerketen wordt maximaal de deductieve afsluiting van u_0 berekend.

Indien een redeneerstap een niet-toegelaten DB-toestand oplevert, dan mag die redeneerstap niet worden gedaan. In het vervolg wordt aangenomen, dat het controle mechanisme CM ook nagaat, of bij een redeneerstap een toegelaten toestandsovergang wordt gedaan. Over de

consequenties van mogelijke redeneringen die niet-toegelaten DB-toestanden opleveren, worden hier geen uitspraken gedaan (zie (Schuwer en Pels, 1991) voor een uitgebreide behandeling hiervan).

Een redeneerprocedure (dat verderop zal worden gedefinieerd) zal een redeneerketen genereren voor het beantwoorden van een query. In feite geeft de bovenstaande definitie van een redeneerketen aan, waaraan een redeneerproces moet voldoen. Het geeft echter niet aan, op welke wijze een redeneerstap wordt gemaakt. Informeel gesteld zal voor het maken van een redeneerstap een hypothese ter beantwoording moeten worden geselecteerd uit de query. Vervolgens is het mogelijk, dat een regel uit de regelbank moet worden geselecteerd voor het uitbreiden van de DB-toestand met afgeleide feiten. Formeel wordt dit als volgt gedefinieerd.

DEFINITIE
De verzameling van redeneertoestanden zal als volgt worden genoteerd:

$$RT = \{ (u,g) \mid u \in U, g \text{ is een query} \}$$

□

DEFINITIE
Een *hypothese-selectiefunctie* is een functie γ met domein RT en $\gamma(u,g) \in g$

Ofwel: door een hypothese-selectiefunctie wordt een hypothese ter beantwoording uit g geselecteerd.
□

DEFINITIE
Een *regel-selectiefunctie* is een functie

$$\rho : RT \rightarrow R$$

Ofwel: door een regel-selectiefunctie wordt een regel uit R geselecteerd om een redeneerstap mee te maken.
□

De invulling van een redeneerstap met behulp van de voorgaande typen functies gebeurt door middel van een redeneerregel. Deze zal op een zodanige wijze een redeneerstap moeten maken, dat een redeneerketen ontstaat, die aan alle eisen van een redeneerketen voldoet. Hiertoe zal een redeneerregel formeel moeten worden gedefinieerd.

DEFINITIE
Stel (u,g) een redeneertoestand. Een *redeneerregel* is een relatie

$$i : RT \rightarrow RT$$

zodat de overgang van (u,g) naar i((u,g)) een redeneerstap is.

De verzameling redeneerregels zal worden aangeduid met I.
□

Uit de definitie van redeneerketen kan worden gehaald, dat er vier redeneerregels zijn te formuleren:

1 $i((u,g)) = (u \cup \{A\vartheta\},g)$, $A\vartheta \in k_R(u)\backslash u$, $(L_1,..,L_n \rightarrow A) = \rho((u,g))$, $\{L_1\vartheta,..,L_n\vartheta\} \subseteq u$.
2 $i((u,g)) = (u,(g\backslash\{A\})\vartheta)$, $A = \gamma((u,g))$, $A\vartheta \in u$ of $(\sim A \in u)$.
3 $i((u,g)) = (u \cup \{A\vartheta\},(g\backslash\{A\})\vartheta)$, $A \in \gamma((u,g))$, $A\vartheta \in k_R(u)\backslash u$, $(L_1,..,L_n \rightarrow A) = \rho((u,g))$, $\{L_1\vartheta,..,L_n\vartheta\} \subseteq u$.
4 $i((u,g)) = (u,(g \cup \{L_1,..,L_n\})\vartheta)$, $(L_1,..,L_n \rightarrow A) = \rho((u,g))$, $A = \gamma((u,g))$, $A\vartheta \in g$.

Hierbij is steeds ϑ een grondsubstitutie.

In het vervolg zullen de redeneerregels voor geval 1 t/m 4 worden aangeduid met resp. i_1, i_2, i_3, i_4.

Regel i_1 beschrijft de redeneerstap, waarbij een afgeleid feit aan de DB-toestand wordt toegevoegd. Deze regel voldoet aan eis a en eis b_1 uit de definitie van redeneerketen. Regel i_2 beschrijft de redeneerstap, waarbij een beantwoorde hypothese wordt verwijderd uit de query, omdat de hypothese of zijn tegengestelde reeds als feit voorkwam in de DB-toestand. Deze regel voldoet aan eis a en eis b_2. Bij regel i_3 wordt een beantwoorde hypothese uit de query verwijderd en als afgeleid feit toegevoegd aan de DB-toestand. Deze regel is een mengvorm van de voorgaande twee regels. Regel i_4 tenslotte beschrijft de situatie, waarin een aantal hypothesen aan de query wordt toegevoegd zodanig, dat beantwoording van die hypothesen tevens beantwoording van een reeds in de query aanwezige hypothese inhoudt. Deze regel voldoet aan de eisen a en b_3.

Merk op, dat de redeneerregels expliciet gegeven worden, terwijl voor de regel-selectiefuncties en de hypothese-selectiefuncties alleen het domein en het bereik worden gegeven, maar niet de invulling. Bij implementaties van deze functies en de redeneerregels bestaan

vrijheden. Bekend zijn de gevallen, waarbij alleen de redeneerregels $\{i_1, i_2, i_3\}$ of $\{i_1, i_2, i_4\}$ geïmplementeerd worden. In het eerste geval wordt de wijze van redeneren *forward chaining* genoemd, in het laatste geval *backward chaining*. Het is aannemelijk te maken, dat het juist de invulling van de regel-selectiefuncties en de hypothese-selectiefuncties zijn, die bepalend zijn voor de kwaliteit van de implementatie. Hierbij wordt onder kwaliteit van een implementatie verstaan de wijze, waarop een redeneerketen wordt opgebouwd voor het beantwoorden van een query. Zo is in hoofdstuk 4 al aangegeven, dat een "triviale" implementatie van beide functies een aantal problemen veroorzaakt met betrekking tot inzichtelijkheid van het systeem en snelheid van redenering.

Gegeven een kennisbank <u,R> moet nog worden aangetoond, dat het mogelijk is, om met behulp van de vier redeneerregels inderdaad de deductieve afsluiting van "u" onder R te berekenen. De volgende stelling toont dit aan.

STELLING A.4
Stel gegeven een kennisbank <u,R>. Stel de regel-selectiefunctie zodanig, dat alle regels uit R ooit kunnen worden gekozen. Dan kan met behulp van alleen redeneerregel i_1 $k_R(u)$ worden berekend.

Bewijs
Volgt rechtstreeks uit de definities van i_1 en deductieve afsluiting.
□

De eis, die aan een regel-selectiefunctie wordt opgelegd in bovenstaande stelling, is een belangrijke eis. In het vervolg wordt ervan uitgegaan, dat een regel-selectiefunctie inderdaad deze eigenschap bezit. Eveneens zal van de hypothese-selectiefunctie worden geëist, dat alle hypothesen gekozen kunnen worden.

Er wordt steeds ervan uitgegaan, dat de database "u" expliciet aanwezig is. In de praktijk komt het echter nogal eens voor, dat de database eerst "gevuld" moet worden, voordat een redeneerketen kan worden opgebouwd. Dit kan gebeuren door het invullen van een invoerscherm of doordat het systeem tijdens een redenering aanvullende gegevens vraagt aan de gebruiker. Dit type systeemactiviteiten, die tot doel hebben een database op te bouwen of te completeren, wordt *niet* beschouwd als een redeneerstap.

A.4 DEFINITIE VAN EEN KENNISSYSTEEM

Het is nu mogelijk om een eerste, voorlopige beschrijving van een kennissysteem te geven. Gegeven een kennisbank <u,R> en een query g op die kennisbank is een kennissysteem een computerprogramma voor het beantwoorden van die query. Het zal hiervoor het bereik van kennis van de kennisbank berekenen. Voor deze berekening wordt een redeneerketen opgebouwd met als doel de query leeg te maken. Het opbouwen van een redeneerketen gebeurt door middel van een redeneerprocedure. Deze procedure maakt gebruik van een controle mechanisme, regel-selectiefunctie ρ, hypothese-selectiefunctie γ en redeneerregels. De laatste drie typen functies en regels zullen in het vervolg *metaregels* worden genoemd.

In paragraaf A.4.1 zal de definitie worden ontwikkeld voor de situatie, waarbij geen aparte afspraken worden gemaakt voor ontoegankelijke uitspraken. Paragraaf A.4.2 behandelt de situatie, waarin die aparte afspraken wel worden gemaakt. Een kennissysteem waar geen afspraken voor ontoegankelijke uitspraken worden gebruikt, wordt verder *strikt redenerend* genoemd. Het andere type kennissysteem wordt *uitgebreid redenerend* genoemd.

A.4.1 STRIKT REDENEREN

DEFINITIE
Een *redeneerprocedure* IP is een computerprogramma, dat door de volgende pre- en postcondities wordt beschreven:

$\{ \quad ((u,R) \text{ is een kennisbank}) \wedge (g = \{ L_1,..,L_n \} \text{ is een query}) \wedge u_0 = u \wedge g_0 = g \}$
$IP(CM,I,\gamma,\rho)$
$\{ (\exists n \in \mathbb{N}: <(u_0,g_0),..,(u_n,g_n)> \text{ is een redeneerketen}) \wedge$
$(u_n = k_R(u_0) \vee g_n = \varnothing \vee (\exists i: \sim L_i \in u_n)) \wedge$
$((u_n = k_R(u_0) \wedge g_n \neq \varnothing \wedge g_0 \text{ is grondquery}) \Leftrightarrow (\forall i: L_i \in g_n: L_i \notin u_n \wedge \sim L_i \notin u_n)) \wedge$
$((g_0 \text{ is geen grondquery}) \Rightarrow u_n = k_R(u_0)) \}$
□

Het resultaat van een redeneerprocedure is een redeneerketen. De procedure kan op drie manieren eindigen:

1 De gehele deductieve afsluiting is berekend (eventueel voordat alle hypothesen uit de query zijn beantwoord)

2 Alle hypothesen uit de query zijn beantwoord (eventueel voordat de gehele deductieve afsluiting is berekend)
3 Er is van een hypothese vastgesteld, dat zijn tegengestelde reeds aanwezig is binnen de DB-toestand.

Tevens zijn, na berekening van de gehele deductieve afsluiting, alle hypothesen uit de query verwijderd, die (eventueel na een grondsubstitutie) element zijn van de deductieve afsluiting. Dit betekent dat, na berekening van de deductieve afsluiting, als g nog hypothesen bevat, dit grondhypothesen zijn. Verder wordt voor een niet-grondquery de gehele deductieve afsluiting berekend voor het geven van een antwoord: er wordt dan immers een enumeratie verwacht van alle mogelijke grondsubstituties in de query, die elementen uit de deductieve afsluiting opleveren. Kort gezegd wordt door de postconditie van de procedure de situatie voorgesteld, dat een antwoord op de query kan worden gegeven of dat er geen redeneerstappen meer kunnen worden gedaan.

Opgemerkt dient te worden, dat de hiervoor gegeven definitie van redeneerprocedure een theoretische definitie is. Zo kan een praktische implementatie van zo'n procedure zodanig zijn gemaakt, dat niet de gehele deductieve afsluiting wordt berekend in het geval van een niet-grondquery. De procedure "weet" dan echter wel, dat verder rekenen geen gegevens oplevert die voor het geven van het antwoord benodigd zijn. Dit is theoretisch gelijkwaardig met het berekenen van de gehele deductieve afsluiting.

Doordat functiesymbolen worden toegestaan, is het in theorie mogelijk, dat een oneindig lange redeneerketen ontstaat (Lloyd, 1987). Dit geval zal verder niet worden beschouwd. Er wordt dus uitgegaan van het zodanig gebruiken van functiesymbolen, dat deze situatie niet kan ontstaan. Anders gesteld: evaluatie van functies is beslisbaar in eindige tijd.

Eerder werd een theoretische antwoordfunctie gedefinieerd. Vanuit de postconditie van de redeneerprocedure kan worden aangegeven, welk antwoord op een query zal moeten worden gegeven, opdat dit overeenstemt met het theoretische antwoord. Dit antwoord op de query moet worden afgelezen uit de inhoud van de eindtoestanden van zowel de database als de verzameling hypothesen. De volgende definitie geeft het verband weer tussen die eindtoestanden en het te geven antwoord.

DEFINITIE

Stel (u,R) een kennisbank, g = $\{L_1,..,L_k\}$ een query. Stel IP een redeneerprocedure, die een eindtoestand (u_n,g_n) genereert. Een *antwoord op een query g* is een functie

$$ANT : g \rightarrow \{TRUE, FALSE, UNKNOWN\}$$

die als volgt wordt gedefinieerd:

A. Stel alle L_i zijn gronduitspraken.

1 $\forall i: L_i \in u_n \wedge g_n = \varnothing \Rightarrow ANT(g) = TRUE$

2 $\exists i: \sim L_i \in u_n \Rightarrow ANT(g) = FALSE$

3 $\exists L_j: \sim L_j \notin u_n \wedge L_j \notin u_n \Rightarrow ANT(g) = UNKNOWN$

B. Stel niet alle L_i gronduitspraken. Stel $x_1,..,x_s$ de variabelen.
Definieer $\Theta = \{\vartheta \mid \vartheta = \{x_1\backslash c_1,..,x_s\backslash c_s\}$ een grondsubstitutie, $\{L_1\vartheta,..,L_k\vartheta\} \subseteq u_n\}$.
Dan $ANT(g) = \Theta$
□

Er moet natuurlijk wel worden aangetoond, dat dit antwoord op een query overeenstemt met het theoretische antwoord.

STELLING A.5

Stel (u,R) een kennisbank, g = $\{L_1,..,L_k\}$ een query. Stel IP een redeneerprocedure, die een eindtoestand (u_n,g_n) genereert. Stel ANT(g) het antwoord op de query. Stel a de theoretische antwoordfunctie, zoals gedefinieerd in paragraaf A.3. Dan geldt: ANT(g) = a(g).

Bewijs
Stel os = $\{g_1',..,g_k'\}$ een orthogonale splitsing van g.

I. Stel k = 1.
A1.
$((\forall i: L_i \in u_n) \wedge (g_n = \varnothing) \wedge (u_n \subseteq k_R(u)$ (uit de definitie van redeneerketen))) $\Rightarrow$
$(\forall i: L_i \in k_R(u)) \Rightarrow$ (uit de definitie van antwoord) a(g) = TRUE.
Dus a(g) = ANT(g).
A2.
$(\exists i: \sim L_i \in u_n) \Rightarrow (\exists i: \sim L_i \in k_R(u)) \Rightarrow a(g) = FALSE$. Dus a(g) = ANT(g).

A3.
($\exists j$: $\sim L_j \notin u_n \wedge L_j \notin u_n$) $\Rightarrow$ ((Uit de postconditie van IP) $u_n = k_R(u) \wedge L_j \notin u_n$)
$\Rightarrow$ a(g) = UNKNOWN. Dus a(g) = ANT(g).
B1.
Uit de postconditie van IP is direct te halen, dat:
$\Theta = \{\vartheta \mid \vartheta = \{x_1\backslash c_1,..,x_s\backslash c_s\}$ een grondsubstitutie, $\{L_1\vartheta,..,L_k\vartheta\} \subseteq k_R(u)\}$. Hieruit volgt:
a(g) = Θ, dus a(g) = ANT(g).

II Stel k = 2.
Omdat de stelling al geldt voor ANT(g_1') en ANT(g_2'), en de tabel bij de theoretische antwoordfunctie welgedefinieerd is en ook hier wordt gebruikt voor het berekenen van het totale antwoord, geldt de stelling ook voor g.

III Stel k > 2.
De juistheid van de stelling volgt door het herhaald toepassen van stap II.

QED
□

Het volgende voorbeeld laat zien, dat verschillende redeneerprocedures op verschillende manieren tot een antwoord voor een query kunnen komen.

VOORBEELD A.10
Stel gegeven de DB-toestand u_0 = { ouder(claus,johan-friso), man(claus) }, de regelbank R = { ouder(x,y) $\wedge$ man(x) $\rightarrow$ vader(x,y) } en de volgende verzameling redeneerregels:

I = { i_1,i_2,i_3,i_4 }

Beschouw de query g_0 = { vader(claus,y) }. De volgende drie redeneerketens kunnen door drie verschillende redeneerprocedures (die alle gebruik maken van de verzameling I) worden opgebouwd (hierbij refereert ϑ steeds naar de grondsubstitutie, die bij de diverse redeneerregels wordt genoemd):

a.

ϑ = {x\claus} $i_4(u_0,g_0)$ $= (u_1,g_1)$
$= (u_0;g_0 \cup \{$ ouder(claus,y), man(claus) $\})$

ϑ = {y\johan-friso} $i_2(u_1,g_1)$ $= (u_2,g_2)$
$= (u_0,\{$ man(claus), vader(claus,johan-friso)$\})$

$\vartheta = \varnothing$ $i_2(u_2,g_2)$ $= (u_3,g_3)$
$= (u_0,\{$ vader(claus,johan-friso) $\})$

ϑ = {x\claus,y\johan-friso}
$i_3(u_3,g_3)$ $= (u_0 \cup \{$ vader(claus,johan-friso)$\}, \varnothing)$

b.

ϑ = {x\claus,y\johan-friso}
$i_3(u_0,g_0)$ $= (u_0 \cup \{$ vader(claus,johan-friso)$\}, \varnothing)$

c.

ϑ = {x\claus,y\johan-friso}
$i_1(u_0,g_0)$ $= (u_1,g_1)$
$= (u_0 \cup \{$ vader(claus,johan-friso)$\}, g_0)$

ϑ = {y\johan-friso} $i_2(u_1,g_1)$ $= (u_1, \varnothing)$

In alle gevallen is de waarde van het antwoord op de query:

$a(g_0)$ = { y\johan-friso }

In alle gevallen is ook de gehele deductieve afsluiting van u_0 berekend.
□

De redeneerketen, die door een redeneerprocedure wordt gegenereerd ter beantwoording van een query g, kan worden beschouwd als een *bewijs* voor het antwoord. Dit bewijs hoeft niet hetzelfde te zijn als de wijze van redeneren, die door het systeem wordt gevolgd. Zo is het bijvoorbeeld mogelijk dat, door een bepaalde keuze van een regel in een bepaalde redeneertoestand, het redeneerproces geen resultaat oplevert. Door terug te gaan naar de redeneertoestand, waar de "verkeerde" regel werd gekozen en daar met een andere regel verder te gaan, kan geprobeerd worden wel tot een antwoord te komen. Dit *backtrackingmechanisme* is niet terug te vinden in de uiteindelijk gegenereerde redeneerketen: het is immers bij het geven van het antwoord niet meer van belang, welke beslissingen tijdens het redeneerproces niet bijgedragen hebben tot het resultaat. Een (theoretische) consequentie van deze stellingname is, dat alle feiten, die gegenereerd werden gedurende het doodlopende deel van het redeneer-

proces, weer uit de database verwijderd worden bij het backtracken. Als sommige van die feiten verderop in het redeneerproces weer benodigd zijn, kunnen ze alsnog worden afgeleid en is die afleiding ook terug te vinden in de door de redeneerprocedure gegenereerde redeneerketen.

De volgende stelling toont aan, dat het niet toevallig is, dat in het voorbeeld alle redeneerprocedures uiteindelijk hetzelfde antwoord genereren.

STELLING A.6
Stel gegeven een kennisbank <u,R> en een query g = { L_1,..,L_k} . Stel gegeven een redeneerprocedure IP(CM,I,γ,ρ), die een redeneerketen <(u,g),(u_1,g_1),..,(u_n,g_n)> genereert.
Stel a = ANT(g) het antwoord op de query. Stel IP'(CM',I',γ',ρ') een andere redeneerprocedure, die een redeneerketen <(u,g),(u_1',g_1'),..,(u_m',g_m')> genereert. Stel hierbij a' = ANT'(g) het antwoord. Dan geldt a = a'.

Bewijs
Het is voldoende alleen het geval te beschouwen, dat g een orthogonale splitsing is van zichzelf.

Onderscheid de volgende gevallen.

g bevat alleen grondhypothesen
Omdat g zijn eigen orthogonale splitsing is, geldt, dat g = { L_1 }. Onderscheid de volgende gevallen:

a. $L_1 \in k_R(u)$. Uit de postcondities van IP en IP'volgt: ($L_1 \in u_n \wedge L_1 \in u_m$') en ($g_n = \emptyset \wedge g_m' = \emptyset$). Uit de definities van a en a' volgt dan direct: a = TRUE en a' = TRUE.

b. $\sim L_1 \in k_R(u)$. Dit wordt vanuit het ongerijmde bewezen. Omdat $u_n \subseteq k_R(u)$, geldt $L_1 \notin u_n$ en (analoog) $L_1 \notin u_m$.

Stel $\sim L_1 \notin u_n$. Dan (postconditie IP) $u_n = k_R(u)$ of $g_n = \emptyset$. Als $u_n = k_R(u)$, dan is er een tegenspraak. Als $g_n = \emptyset$, dan is er een redeneerstap geweest, waarbij L_1 uit g_n is verwijderd. Omdat $L_1 \notin u_n$ en $L_1 \notin k_R(u)$, moet bij die redeneerstap redeneerregel i_2 zijn gebruikt (dit volgt uit de voorwaarden voor het toepassen van de redeneerregels). Maar uit het voorgaande en de voorwaarden voor i_2 volgt, dat $\sim L_1 \in u_n$: tegenspraak.

Dus $\sim L_1 \in u_n$ en (analoog) $\sim L_1 \in u_m$'. Uit de definities van a en a' volgt dan direct: a = FALSE en a' = FALSE.

c. $L_1 \notin k_R(u)$ en $\sim L_1 \notin k_R(u)$. Uit de postconditie van IP en IP' volgt: $g_n \notin \emptyset$ en $g_m' \notin \emptyset$. Dus $L_1 \in g_n$ en $L_1 \in g_m'$. Uit de definitie van a en a' volgt: a = UNKNOWN en a' = UNKNOWN.

g bevat niet-grondhypothesen
Uit de postconditie van IP en IP' volgt, dat $u_n = k_R(u)$ en $u_m' = k_R(u)$. Uit de definitie van a en a' volgt onmiddellijk, dat a(g) = a'(g).
QED
□

A.4.2 UITGEBREID REDENEREN

In de situatie, dat een grondhypothese L behoort tot de verzameling ontoegankelijke uitspraken werd tot nu toe door een antwoordfunctie het antwoord UNKNOWN gegeven. In de praktijk worden vaak afspraken gemaakt over sommige van deze situaties, waardoor een ander antwoord dan UNKNOWN kan worden verkregen. Formeel levert dit de volgende metaregel op (waaarbij O(u,R) eerder werd gedefinieerd als de verzameling ontoegankelijke uitspraken bij een kennisbank <u,R>):

DEFINITIE
Stel gegeven een kennisbank <u,R>. Een *externe regel* is een (totale) functie:

$$e: O(u,R) \rightarrow \{TRUE, FALSE, UNKNOWN\}$$

Verder wordt voor een externe regel e de verzameling $O_e(u,R)$ gedefinieerd als volgt:

$$O_e(u,R) = \{ L \in O(u,R) \mid e(L) = TRUE \}$$

Gegeven een kennisbank <u,R> bevat $O_e(u,R)$ dus die ontoegankelijke uitspraken, die een waarheidswaarde TRUE hebben onder de externe regel e.

Voor een verzameling $D = \{L_1,..,L_k\}$ van ontoegankelijke uitspraken wordt met de (slordige) notatie e(D) bedoeld:

$$e(D) = e(L_1) \wedge e(L_2) \wedge ... \wedge e(L_k)$$

(met TRUE $\wedge$ UNKNOWN $\equiv$ UNKNOWN en FALSE $\wedge$ UNKNOWN $\equiv$ FALSE).
□

VOORBEELD A.11
Voor een kennisbank <u,R> is de bekende Closed World Assumption (CWA) een externe regel:

e_{CWA}: O(u,R) → { FALSE }
□

VOORBEELD A.12
Voor een kennisbank <u,R> is de Negation as finite failure-regel (NFF-regel) uit Prolog als volgt te formuleren als externe regel:

e_{NFF}: O(u,R) → {TRUE, FALSE, UNKNOWN}, gedefinieerd als:

$e_{NFF}(L)$ = TRUE	voor L een negatieve uitspraak, waarbij alle eindige redeneringen om ~L te bewijzen, falen
$e_{NFF}(L)$ = FALSE	voor L een negatieve uitspraak, waarbij een eindige redenering om ~L te bewijzen, succes heeft
$e_{NFF}(L)$ = UNKNOWN	anders

□

Externe regels zijn alleen gedefinieerd voor ontoegankelijke grondhypothesen. Dit betekent, dat voor een niet-grondhypothese een externe regel pas gebruikt kan worden, als eerst een grondsubstitutie heeft plaatsgevonden en dan blijkt, dat de op die wijze verkregen grondhypothese ontoegankelijk is.

Het gebruik van een externe regel heeft gevolgen voor eerder gegeven definities van antwoordfunctie, redeneerketen, redeneerregels, en inferentieprocedure. Met behulp van een externe regel is het namelijk mogelijk feiten af te leiden, die geen element zijn van de deductieve afsluiting van de kennisbank. Formeel wordt dit vastgelegd in de volgende begrippen. Hierbij zal dezelfde betooglijn worden gevolgd als in paragraaf A.4.1.

DEFINITIE
Stel <u,R> een kennisbank. Stel $g=\{L_1,..,L_n\}$ een query. Stel os = $\{g_1,..,g_k\}$ een orthogonale splitsing van g. Stel a de theoretische antwoordfunctie voor g. Stel e een externe regel. Een *uitgebreid antwoord op een query* is een functie a', die als volgt wordt gedefinieerd.

A. Stel k = 1. Onderscheid:

1. Alle L_i's zijn grondhypothesen. Dan g = $\{L_1\}$. Dan:

   a': $\wp L(S) \rightarrow$ { TRUE, FALSE, UNKNOWN }. Meer specifiek:

   - a'(g) = a(g) als $L_1 \notin O(u,R)$
   - a'(g) = e(g) als $L_1 \in O(u,R)$

   (Respectievelijk: voor ontoegankelijke uitspraken wordt het uitgebreide antwoord gegeven door de externe regel. In alle andere gevallen komt het uitgebreide antwoord overeen met het eerder gedefinieerde antwoord.)

2. Veronderstel dat binnen de query de variabelen $x_1,..,x_s$ voorkomen.

   Dan: a'(g) = Θ, waarbij Θ is gedefinieerd als:

   $\Theta = \{ \vartheta \mid \vartheta=\{x_1\backslash c_1,..,x_s\backslash c_s\}$ een grondsubstitutie,

   $\{L_1\vartheta,..,L_n\vartheta\} \subseteq k_R(u) \cup O_e(u,R) \}$

   (Ofwel: het antwoord op de query bestaat uit alle mogelijke substituties binnen de hypothesen, zodat de query een deelverzameling wordt van de deductieve afsluiting van <u,R> verenigd met de verzameling van ontoegankelijke uitspraken, die waarheidswaarde TRUE krijgt door toepassen van de externe regel.)

B. Stel k = 2. Stel a'(g_j) het antwoord voor g_j (j = 1,2).
Dan is het antwoord a'(g) uit de volgende tabel af te lezen.

a'(g_2)↓ a'(g_1)→	TRUE	FALSE	e(g_1)	Θ
TRUE	TRUE	FALSE	e(g_1)	Θ
FALSE	FALSE	FALSE	FALSE	$\varnothing$
e(g_2)	e(g_2)	FALSE	e($g_1 \cup g_2$)	Θ_2
Θ'	Θ'	$\varnothing$	Θ_1	$\Theta \otimes \Theta$'

Met Θ_1 = { $\vartheta \mid \vartheta \in \Theta' \wedge (\forall L: L \in g_1: e(L\vartheta) = TRUE)$ } en
Θ_2 = { $\vartheta \mid \vartheta \in \Theta \wedge (\forall L: L \in g_2: e(L\vartheta) = TRUE)$ }.
(Merk op: $\Theta_1 \neq \varnothing \Leftrightarrow e(g_1) = TRUE$, omdat g_1 alleen grondhypothesen bevat. Analoog voor Θ_2.)

C. Stel k > 2. Dan is a'(g) te verkrijgen door herhaald toepassen van de tabel.

In het vervolg zal a' de uitgebreide theoretische antwoordfunctie worden genoemd.
□

Merk op, dat de definitie van uitgebreid antwoord volledig overeenstemt met de definitie van antwoord in het geval, dat er geen externe regel aanwezig is. In feite is die situatie te vergelijken met de situatie, waarbij een externe regel aan alle hypothesen uit O(u,R) de waarde UNKNOWN geeft.

DEFINITIE
Stel e een externe regel. Een *uitgebreide redeneerketen* is een rij $<(u_0,g_0),..,(u_k,g_k)>$ van redeneertoestanden zodanig dat het volgende geldt:

(u_i,g_i) is een redeneertoestand (alle i).
Voor alle i, $1 \leq i \leq k$ geldt:

a. $u_{i-1} \subseteq u_i \subseteq k_R(u_{i-1}) \cup O_e(u_{i-1},R)$

Een van de volgende drie gevallen geldt:

b_1. $g_i = g_{i-1}$
b_2. $g_i = (g_{i-1} \backslash L)\vartheta$, $L\vartheta \in k_R(u_0) \wedge O_e(u_{i-1},R)$, $L \in g_{i-1}$.
b_3. $g_i = (g_{i-1} \cup \{L_1,..,L_n\})\vartheta$, $L_1,..,L_n \rightarrow A \in R$, $A\vartheta \in g_{i-1}$

c. $(u_{i-1},g_{i-1}) \neq (u_i,g_i)$

(Met ϑ een grondsubstitutie).

De verzameling van uitgebreide redeneerketens zal worden genoteerd als RTU.
□

(Reeds eerder werd aangetoond, dat de eerste eis kan worden vervangen door:
voor alle i, $1 \leq i \leq k$ geldt: $u_{i-1} \subseteq u_i \subseteq k_R(u_0) \cup O_e(u_{i-1},R)$). Het is eenvoudig in te zien, dat RT $\subseteq$ RTU (beschouw RT als het geval, waarbij de externe regel alleen de waarde UNKNOWN aan hypothesen uit $O(u_k,R)$ geeft. Er geldt dan $O_e(u_k,R) = \varnothing$).

DEFINITIE
Stel e een externe regel. Een *uitgebreide redeneerregel* is een relatie

$i: RTU \rightarrow RTU$

die als volgt wordt gedefinieerd:

a. $i((u,g)) \in \{ i_1((u,g)), i_2((u,g)), i_3((u,g)), i_4((u,g)) \}$ (met $i_1,..,i_4$ de redeneerregels, zoals eerder gedefinieerd).
b. $i((u,g)) = (u \cup \{A\vartheta\},(g\backslash\{A\})\vartheta)$, $A \in \gamma((u,g))$, $A\vartheta \in O_e(u,R)$

Voortaan zal de redeneerregel van b. (waarmee feiten aan de database worden toegevoegd, die een waarde TRUE krijgen via de externe regel) worden aangeduid als i_e. De verzameling van uitgebreide redeneerregels zal worden genoteerd als I_e.
□

Het is eenvoudig in te zien, dat de uitgebreide redeneerregel i_e een correcte redeneerstap genereert.

DEFINITIE
Een *uitgebreide redeneerprocedure* IPU is een computerprogramma, dat door de volgende pre- en postcondities is bepaald:

{ ((u,R) is een kennisbank) ∧
(g = { $L_1,..,L_n$ } is een query) ∧
$u_0 = u \wedge g_0 = g$ }
IPU(CM,e,I_e,γ,ρ)
{ $(\exists n \in \mathbb{N}: <(u_0,g_0),..,(u_n,g_n)>$ is een uitgebreide redeneerketen) ∧
$(u_n = k_R(u_0) \cup O_e(u_n,R) \vee g_n = \varnothing \vee (\exists i: \sim L_i \in u_n)) \wedge$
$((u_n = k_R(u_0) \cup O_e(u_n,R) \wedge g_n \neq \varnothing \wedge g_0$ is een grondquery) ⇔
$(\forall i: L_i \in g_n: L_i \in O(u_n,R) \setminus O_e(u_n,R))) \wedge$
$((g_0$ is geen grondquery) ⇒ $u_n = k_R(u_0) \cup O_e(u_n,R))$ }
□

Door een uitgebreide redeneerprocedure worden ook feiten die waarheidswaarde TRUE krijgen door middel van een externe regel toegevoegd aan de database. Het is (door bestudering van de postcondities) makkelijk na te gaan, dat in de situatie met $O_e(u_n,R) = \varnothing$

een uitgebreide redeneerprocedure dezelfde resultaten oplevert als een "gewone" redeneerprocedure.

DEFINITIE
Stel (u,R) een kennisbank, e een externe regel en $g = \{L_1,..,L_k\}$ een query. Stel IPU een uitgebreide redeneerprocedure, die een eindtoestand (u_n,g_n) genereert. Het *antwoord op de query g* is een functie

$$ANT: g \rightarrow \{TRUE, FALSE, UNKNOWN\}$$

die als volgt wordt gedefinieerd:

A. Stel alle L_i zijn gronduitspraken.
1 $\forall i: L_i \in u_n \wedge g_n = \varnothing \Rightarrow ANT(g) = TRUE$
2 $\exists i: \sim L_i \in u_n \Rightarrow ANT(g) = FALSE$
3 $\forall j: \sim L_j \notin u_n \wedge L_j \notin u_n \Rightarrow ANT(g) = e(g_n)$

B. Stel niet alle L_i gronduitspraken. Stel $x_1,..,x_s$ de variabelen.
Definieer $\Theta = \{\vartheta \mid \vartheta = \{x_1\backslash c_1,..,x_s\backslash c_s\}$ een grondsubstitutie, $\{L_1\vartheta,..,L_k\vartheta\} \subseteq u_n\}$.
Dan $ANT(g) = \Theta$
□

STELLING A.7
Stel (u,R) een kennisbank, e een externe regel en $g = \{L_1,..,L_k\}$ een query. Stel IPU een uitgebreide redeneerprocedure, die een eindtoestand (u_n,g_n) genereert. Stel ANT(g) het antwoord op de query. Stel a' de uitgebreide theoretische antwoordfunctie. Dan geldt: $ANT(g) = a'(g)$.

Bewijs
Analoog aan het bewijs van stelling A.5. In geval A3 moet worden opgemerkt, dat $g_n \subseteq g$ en dat $g\backslash g_n \cap O(u,R) = \varnothing$: g_n bevat alleen die (grond)hypothesen uit O(u,R), die geen waarheidswaarde TRUE hebben.
□

In stelling A.6 is bewezen, dat *onafhankelijk van de inhoud van I,* γ *en* ρ, voor een query door twee verschillende redeneerprocedures IP en IP' altijd hetzelfde antwoord wordt gegenereerd. De enige eis, die (in een eerder stadium) aan γ en ρ is opgelegd is, dat door middel van de regel-selectiefuncties alle regels uit R resp. door middel van de hypothese-

selectiefuncties alle hypothesen uit de query kunnen worden geselecteerd. In de situatie, waarbij voor de beantwoording van een query gebruik wordt gemaakt van een externe regel (en dus van een uitgebreide redeneerprocedure), geldt deze stelling alleen nog maar als de beide redeneerprocedures dezelfde externe regel gebruiken. Het belangrijke verschil is dus gelegen in het feit, dat eenduidigheid van antwoord afhangt van *de inhoud van de externe regel.* Merk in dit verband ook op, dat door de postconditie van de (uitgebreide) redeneerprocedure de situatie vermeden wordt, dat (bijvoorbeeld door een niet adequaat regel-selectiemechanisme) de procedure niet in staat is de deductieve afsluiting te berekenen. Als deze situatie zich in de praktijk voordoet, voldoet de redeneerprocedure niet aan de eisen, die er door de pre- en postconditie aan worden opgelegd.

In het voorgaande werd de dynamiek van een kennissysteem beschreven. De formele definitie van een kennissysteem is nu als volgt:

DEFINITIE
Stel gegeven een databaseskelet S en een verzameling constraints C. Een *kennissysteem* is een tupel $<U(S), u, R, CM, e, I_e, \gamma, \rho, IPU(CM,e,I,\gamma,\rho)>$, waarbij

$U(S)$	het toegelaten DB-universum met betrekking tot C
$u \in U(S)$	een toegelaten DB-toestand
R	een regelbank
CM	een controle mechanisme
e	een externe regel
I_e	een verzameling uitgebreide redeneerregels
γ	een hypothese-selectiefunctie
ρ	een regel-selectiefunctie
$IPU(CM,e,I_e,\gamma,\rho)$	een uitgebreide redeneerprocedure

□

TENSLOTTE
Veel van de begrippen en stellingen, die in de opbouw naar de uiteindelijke definitie van een kennissysteem werden geformuleerd en bewezen, zijn ook terug te vinden in (Lloyd, 1987). Lloyd onderkent echter niet de componenten, die in een kennissysteem volgens de bovenstaande definitie worden onderscheiden. Daartegenover staat dat in het betoog in dit hoofdstuk geen aandacht wordt besteed aan aspecten als completeness van de redeneerprocedure en afsluiting van de verzameling regels en feiten, iets waaraan Lloyd wel aandacht besteed. Het doel van deze studie was echter te komen tot een formeel onderbouwde,

duidelijke en werkbare definitie van een kennissysteem. In hoofdstuk 3 en 4 wordt aannemelijk gemaakt, dat de definitie, zoals die uiteindelijk hier wordt gegeven, duidelijk en werkbaar is en daarnaast rijk genoeg om een grote klasse van kennissystemen te beschrijven.

BIJLAGE B. FORMELE DEFINITIE VAN DE DATABASE "BUREAU-EILAND"

NOTATIE

$[a;b] = \{ x \in \mathbb{N} \mid a \leq x \leq b \}$ $(a,b \in \mathbb{N})$

De afkorting "u.i." staat voor "uniek identificerend".

□

DB-UNIVERSUM

Het DB-universum zal worden beschreven op dezelfde wijze als in (De Brock, 1990).

De beschrijving is als volgt opgebouwd. Allereerst zullen enkele verzamelingen worden gedefinieerd, die meerdere malen voorkomen in de beschrijving van het DB-universum. Vervolgens zullen per entiteit achtereenvolgens de objectkarakterisering, de tupelkarakterisering en de tabelkarakterisering worden gegeven. Tenslotte wordt de databasekarakterisering en de definitie van het DB-universum gegeven.

Vooruitlopend op de definitie van de regelbank wordt bij de definitie van het DB-universum al rekening gehouden met keuzevrijheden bij de configuratie van een bureau-eiland. Een minimum configuratie bestaat uit 1 of meer bureaus en een koppeldeel. Ladenblokken, aanhangtafels en stoelen zijn niet verplicht. Voor dit doel worden op de geëigende plaatsen "phantom"-tupels toegevoegd. Dit zijn tupels met een bijzondere sleutel (nl. sleutelwaarde 0), die de niet-keuze van het item representeren.

DEFINITIE VAN VERZAMELINGEN

BKLEURVZ	=	{ 'naturel', 'lichtgrijs', 'grijs', 'zwart', 'bruin' }
KKLEURVZ	=	BKLEURVZ ∪ { 'rood', 'blauw' }
SKLEURVZ	=	{ 'rood', 'grijs', 'blauw', 'bruin', 'groen' }
MATVZ	=	{ 'hout', 'plastic' }
LMAATVZ	=	{180, 200}
BMAATVZ	=	{80, 90}
BLOKSAM	=	{ 'leglade', 'hanglade' }
PLAATSVZ	=	{ 'links', 'rechts' }
BOOLEAN	=	{TRUE, FALSE}

KARAKTERISERINGEN PER ENTITEIT

FBUREAU =
{
(bureaucode, [1000000;9999999]),
(materiaal, MATVZ),
(kleur, BKLEURVZ),
(lengte, LMAATVZ), [R04]
(breedte, BMAATVZ)
}

TBUREAU =
{ t | t ∈ Π(FBUREAU) ∧ (t(materiaal) = 'hout' ⇔ t(kleur) = 'naturel') } [R02]

WBUREAU =
{ T ⊆ TBUREAU | {bureaucode} is u.i. in T }

FLADENBLOK =
{
(ladenblokcode, {0} ∪ [10;99]),
(samenstelling, BLOKSAM ∪ {'NVT'}), [R08]
(materiaal, MATVZ ∪ {'NVT'}),
(kleur, BKLEURVZ ∪ {'NVT'})
}

TLADENBLOK = Π(FLADENBLOK)

WLADENBLOK =
{ T ⊆ TLADENBLOK | {ladenblokcode} is u.i. in T ∧ (∀t ∈ T: t(ladenblokcode) = 0 ⇔
t ↾ {samenstelling, materiaal, kleur} = 'NVT') ∧
(∀t ∈ T: t(materiaal) = 'hout' ⇔ t(kleur) = 'naturel') }

FAANHANG =
{
(aanhangcode, {0} ∪ [100,999]),
(lengte, LMAATVZ ∪ {0}),
(breedte, BMAATVZ ∪ {0}),
(materiaal, MATVZ ∪ {'NVT'}),
(kleur, BKLEURVZ ∪ {'NVT'})
}

TAANHANG = Π(FAANHANG)

WAANHANG =
{ T ⊆ TAANHANG | {aanhangcode} is u.i. in T ∧ (∀t ∈ T: t(aanhangcode) = 0 ⇔
(t ↾ {materiaal, kleur} = 'NVT' ∧ t ↾ {lengte,breedte} = 0)) ∧
(∀t ∈ T: t(materiaal) = 'hout' ⇔ t(kleur) = 'naturel') }

FKOPPEL =
{
(koppelcode, [10;99]),
(kleur, KKLEURVZ), [R11]
(breedte, BMAATVZ),
(materiaal, MATVZ),
(aantal, {1,2,3,4})
}

TKOPPEL = Π(FKOPPEL)

WKOPPEL =
{ T ⊆ TKOPPEL | {koppelcode} is u.i. in T ∧
(∀t ∈ T: t(materiaal) = 'hout' ⇔ t(kleur) = 'naturel') }

FSTOEL =
{
(stoelcode, {0} ∪ [1000,9999]),
(kleur, SKLEURVZ ∪ {'NVT'}), [R24]
(draaibaar, BOOLEAN ∪ {'NVT'}),
(rijdbaar, BOOLEAN ∪ {'NVT'}),
(armleuning, BOOLEAN ∪ {'NVT'}) [R30]
}

TSTOEL = Π(FSTOEL)

WSTOEL =
{ T ⊆ TSTOEL | {stoelcode} is u.i. in T ∧ ∀t ∈ T: t(stoelcode) = 0 ⇔
t ⊦ {kleur,draaibaar,rijdbaar,armleuning} = 'NVT' }

FBURLAD =
{
(bureaucode, [1000000;9999999]),
(ladenblokcode, [10;99]),
(plaats, PLAATSVZ)
}

TBURLAD = Π(FBURLAD)

WBURLAD =
{ T ⊆ TBURLAD | {bureaucode, ladenblokcode, plaats} is u.i. in T }

FBURAANHANG =
{
(bureaucode, [1000000;9999999]),
(aanhangcode, [100;999]),
(bevestiging, PLAATSVZ)
}

TBURAANHANG = Π(FBURAANHANG)

WBURAANHANG =
{ T ⊆ TBURAANHANG | {bureaucode, aanhangcode} is u.i. in T }

FAANHANGLAD =
{
(aanhangcode, [100;999]),
(ladenblokcode, [10;99]),
(plaats, PLAATSVZ)
}

TAANHANGLAD = Π(FAANHANGLAD)

WAANHANGLAD =
{ T ⊆ TAANHANGLAD | {aanhangcode, ladenblokcode} is u.i. in T }

FCONFIG =
{
(eilandcode, [1000000;9999999]),
(bureaucode, [1000000;9999999]),
(koppelcode, [10;99]),
(aanhangcode, {0} ∪ [100;999]),
(ladenblokcode1, {0} ∪ [10;99]),
(ladenblokcode2, {0} ∪ [10;99]),
(stoelcode1, {0} ∪ [1000;9999]),
(stoelcode2, {0} ∪ [1000;9999])
}

TCONFIG =
{t ∈ Π(FCONFIG) | t(aanhangcode) = 0 ∧ t(ladenblokcode1) = 0 ⇒
t(ladenblokcode2) = 0 }

WCONFIG =
{ T ⊆ TCONFIG | {eilandcode} is u.i. in T }

DATABASEKARAKTERISERING

FEILAND =

{

(BUREAU	;	WBUREAU),	Type bureau
(LADENBLOK	;	WLADENBLOK),	Type ladenblok
(AANHANG	;	WAANHANG),	Type aanhangtafel
(KOPPEL	;	WKOPPEL),	Type koppelelement
(STOEL	;	WSTOEL),	Type stoel
(BURLAD	;	WBURLAD),	Plaats ladenblok aan bureau
(BURAANHANG	;	WBURAANHANG),	Plaats aanhangtafel aan bureau
(AANHANGLAD	;	WAANHANGLAD),	Plaats ladenblok aan aanhangtafel
(CONFIG	;	WCONFIG),	Configuratie

}

Als een afkorting worden de volgende vier verzamelingen DB-toestanden gedefinieerd (DB-toestanden, waarin een bureau-eiland met resp. 1, 2, 3 of 4 bureaus is geconfigureerd):

1B = {t ∈ v(CONFIG) | v ∈ Π(FEILAND) ∧ ∀t' ∈ v(KOPPEL):
t(koppelcode) = t'(koppelcode) ⇒
t'(aantal) = 1 }

2B = {t ∈ v(CONFIG) | v ∈ Π(FEILAND) ∧ ∀t' ∈ v(KOPPEL):
t(koppelcode) = t'(koppelcode) ⇒
t'(aantal) = 2 }

3B = {t ∈ v(CONFIG) | v ∈ Π(FEILAND) ∧ ∀t' ∈ v(KOPPEL):
t(koppelcode) = t'(koppelcode) ⇒
t'(aantal) = 3 }

4B = {t ∈ v(CONFIG) | v ∈ Π(FEILAND) ∧ ∀t' ∈ v(KOPPEL):
t(koppelcode) = t'(koppelcode) ⇒
t'(aantal) = 4 }

DATABASE-UNIVERSUM

UEILAND =

{ v | v ∈ Π(FEILAND)

en	id(bureaucode)	verbindt	v(BURLAD)	met	v(BUREAU)
en	id(ladenblokcode)	verbindt	v(BURLAD)	met	v(LADENBLOK)
en	id(bureaucode)	verbindt	v(BURAANHANG)	met	v(BUREAU)
en	id(aanhangcode)	verbindt	v(BURAANHANG)	met	v(AANHANG)
en	id(aanhangcode)	verbindt	v(AANHANGLAD)	met	v(AANHANG)
en	id(ladenblokcode)	verbindt	v(AANHANGLAD)	met	v(LADENBLOK) [R19]
en	id(bureaucode)	verbindt	v(CONFIG)	met	v(BUREAU)
en	id(aanhangcode)	verbindt	v(CONFIG)	met	v(AANHANG)
en	{(ladenblokcode1,ladenblokcode)}	verbindt v(CONFIG) met v(LADENBLOK)			
en	{(ladenblokcode2,ladenblokcode)}	verbindt v(CONFIG) met v(LADENBLOK)			
en	{(stoelcode1,stoelcode)}	verbindt v(CONFIG) met v(STOEL)			
en	{(stoelcode2,stoelcode)}	verbindt v(CONFIG) met v(STOEL)			
en	id(koppelcode)	verbindt	v(CONFIG)	met	v(KOPPEL)

en[∀t ∈ v(CONFIG) ∀t' ∈ v(BUREAU) ∀t" ∈ v(LADENBLOK):
t(ladenblokcode1) ≠ 0 ⇒
t(bureaucode) = t'(bureaucode) ∧ t(ladenblokcode1) = t"(ladenblokcode) ⇒
t'(materiaal) = t"(materiaal)

en ∀t ∈ v(CONFIG) ∀t' ∈ v(LADENBLOK) ∀t" ∈ v(AANHANG):
(t(ladenblokcode1) ≠ 0 ∧ t(aanhangcode) ≠ 0) ⇒
t(ladenblokcode1) = t'(ladenblokcode) ∧ t(aanhangcode) = t"(aanhangcode) ⇒
t'(materiaal) = t"(materiaal)

en ∀t ∈ v(CONFIG) ∀t' ∈ v(LADENBLOK) ∀t" ∈ v(AANHANG):
(t(ladenblokcode2) ≠ 0 ∧ t(aanhangcode) ≠ 0) ⇒
t(ladenblokcode2) = t'(ladenblokcode) ∧ t(aanhangcode) = t"(aanhangcode) ⇒
t'(materiaal) = t"(materiaal)

en ∀t ∈ v(CONFIG) ∀t' ∈ v(AANHANG) ∀t" ∈ v(KOPPEL):
t(aanhangcode) ≠ 0 ⇒
t(aanhangcode) = t'(aanhangcode) ∧ t(koppelcode) = t"(koppelcode) ⇒
t'(materiaal) = t"(materiaal)] [R03]

en $\forall t \in v(CONFIG)\ \forall t' \in v(BURLAD)\ \forall t'' \in v(BURLAD)$:
$(t(ladenblokcode1) \neq 0 \wedge t(ladenblokcode2) \neq 0) \Rightarrow$
$t(bureaucode) = t'(bureaucode) \wedge t(bureaucode) = t''(bureaucode) \wedge$
$t(ladenblokcode1) = t'(ladenblokcode) \wedge t(ladenblokcode2) = t''(ladenblokcode) \Rightarrow$
$t'(plaats) \neq t''(plaats)$ [R07]

en $\forall t \in v(CONFIG)\ \forall t' \in v(BUREAU)\ \forall t'' \in v(KOPPEL)$:
$t(bureaucode) = t'(bureaucode) \wedge t(koppelcode) = t''(koppelcode) \Rightarrow$
$t'(breedte) = t''(breedte)$ [R10]

en[$\forall t \in 1B\ \forall t' \in v(BUREAU)\ \forall t'' \in v(KOPPEL)$:
$t(bureaucode) = t'(bureaucode) \wedge t(koppelcode) = t''(koppelcode) \Rightarrow$
$t'(kleur) = t''(kleur)$

en $\forall t \in v(config) \backslash 1B\ \forall t' \in v(BUREAU)\ \forall t'' \in v(KOPPEL)$:
$t(bureaucode) = t'(bureaucode) \wedge t(koppelcode) = t''(koppelcode) \Rightarrow$
$(t'(kleur) = \text{'naturel'} \Rightarrow t''(kleur) = \text{'naturel'}) \wedge$
$(t'(kleur) = \text{'lichtgrijs'} \Rightarrow t''(kleur) \in \{\text{'lichtgrijs'},\text{'zwart'}\}) \wedge$
$(t'(kleur) = \text{'grijs'} \Rightarrow t''(kleur) \in \{\text{'grijs'},\text{'zwart'},\text{'rood'},\text{'blauw'}\}) \wedge$
$(t'(kleur) = \text{'zwart'} \Rightarrow t''(kleur) \in \{\text{'lichtgrijs'},\text{'grijs'},\text{'zwart'},\text{'rood'},\text{'blauw'}\}) \wedge$
$(t'(kleur) = \text{'bruin'} \Rightarrow t''(kleur) = \text{'bruin'})]$ [R12], [R14]

en[$\forall t \in v(CONFIG)\ \forall t' \in v(BUREAU)\ \forall t'' \in v(LADENBLOK)$:
$t(ladenblokcode1) \neq 0 \Rightarrow$
$t(bureaucode) = t'(bureaucode) \wedge t(ladenblokcode1) = t''(ladenblokcode) \Rightarrow$
$t'(kleur) = t''(kleur)$

en $\forall t \in v(CONFIG)\ \forall t' \in v(BUREAU)\ \forall t'' \in v(LADENBLOK)$:
$t(ladenblokcode2) \neq 0 \Rightarrow$
$t(bureaucode) = t'(bureaucode) \wedge t(ladenblokcode2) = t''(ladenblokcode) \Rightarrow$
$t'(kleur) = t''(kleur)$

en $\forall t \in v(CONFIG)\ \forall t' \in v(BUREAU)\ \forall t'' \in v(AANHANG)$:
$t(aanhangcode) \neq 0 \Rightarrow$
$t(bureaucode) = t'(bureaucode) \wedge t(aanhangcode) = t''(aanhangcode) \Rightarrow$
$t'(kleur) = t''(kleur)]$ [R13]

en $\forall t \in v(BUREAU)$:
$(t(lengte) = 200 \Leftrightarrow \exists t' \in 1B: t'(bureaucode) = t(bureaucode))$ [R15], [R16]

en $\forall t \in v(CONFIG)\ \forall t' \in v(BURAANHANG)\ \forall t'' \in v(AANHANGLAD)$:
$(t(aanhangcode) \neq 0 \wedge t(ladenblokcode2) \neq 0) \Rightarrow$
$t(bureaucode) = t'(bureaucode) \wedge t(aanhangcode) = t'(aanhangcode) \wedge$
$t(aanhangcode) = t''(aanhangcode) \wedge t(ladenblokcode2) = t''(ladenblokcode) \Rightarrow$
$t'(bevestiging) \neq t''(plaats)$ [R20]

en $\forall t \in v(CONFIG)\ \forall t' \in v(BURAANHANG)\ \forall t'' \in v(BURLAD)$:
$(t(aanhangcode) \neq 0 \wedge t(ladenblokcode1) \neq 0) \Rightarrow$
$t(bureaucode) = t'(bureaucode) \wedge t(aanhangcode) = t'(aanhangcode) \wedge$
$t(bureaucode) = t''(bureaucode) \wedge t(ladenblokcode1) = t''(ladenblokcode) \Rightarrow$
$t'(bevestiging) \neq t''(plaats)$ [R21]

en $\forall t \in 4B\ \forall t' \in v(BURAANHANG)$:
$t(aanhangcode) \neq 0 \Rightarrow$
$t(bureaucode) = t'(bureaucode) \wedge t(aanhangcode) = t'(aanhangcode) \Rightarrow$
$t'(bevestiging) = 'L'$ [R23]

en $\forall t \in v(CONFIG)\ \forall t' \in v(STOEL)$:
$t(stoelcode1) \neq 0 \Rightarrow$
$t(stoelcode1) = t'(stoelcode) \Rightarrow$
$(t'(draaibaar) = TRUE \wedge t'(rijdbaar) = TRUE)$ [R26]

en $\forall t \in 1B\ \forall t' \in v(STOEL)$:
$t(stoelcode2) \neq 0 \Rightarrow$
$t(stoelcode2) = t'(stoelcode) \Rightarrow$
$(t'(draaibaar) = FALSE \wedge t'(rijdbaar) = FALSE)$ [R27]

en[$\forall t \in v(CONFIG)\ \forall t' \in v(BUREAU)\ \forall t'' \in v(STOEL)$:
$t(stoelcode1) \neq 0 \Rightarrow$
$t(bureaucode) = t'(bureaucode) \wedge t(stoelcode1) = t''(stoelcode) \Rightarrow$
$(t'(kleur) = \text{'lichtgrijs'} \Rightarrow t''(kleur) \in \{\text{'rood','blauw','groen'}\}) \wedge$
$(t'(kleur) = \text{'grijs'} \Rightarrow t''(kleur) \in \{\text{'rood','blauw','groen'}\}) \wedge$
$(t'(kleur) = \text{'zwart'} \Rightarrow t''(kleur) \in \{\text{'grijs','bruin'}\}) \wedge$
$(t'(kleur) = \text{'bruin'} \Rightarrow t''(kleur) \in \{\text{'grijs','bruin','groen'}\})$

en $\forall t \in v(CONFIG)\ \forall t' \in v(BUREAU)\ \forall t'' \in v(STOEL)$:
$t(stoelcode2) \neq 0 \Rightarrow$
$t(bureaucode) = t'(bureaucode) \wedge t(stoelcode2) = t''(stoelcode) \Rightarrow$
$(t'(kleur) = \text{'lichtgrijs'} \Rightarrow t''(kleur) \in \{\text{'rood','blauw','groen'}\}) \wedge$
$(t'(kleur) = \text{'grijs'} \Rightarrow t''(kleur) \in \{\text{'rood','blauw','groen'}\}) \wedge$
$(t'(kleur) = \text{'zwart'} \Rightarrow t''(kleur) \in \{\text{'grijs','bruin'}\}) \wedge$
$(t'(kleur) = \text{'bruin'} \Rightarrow t''(kleur) \in \{\text{'grijs','bruin','groen'}\})]$ [R29]

}

SAMENVATTING

De eerste kennissystemen werden in de jaren 70 ontwikkeld vanuit een research-omgeving. Uit die tijd stammen de "klassiekers" onder de kennissystemen, zoals XCON, MYCIN en PROSPECTOR. In de jaren 80 kwamen kennissystemen in de belangstelling van het bedrijfsleven. Veel van de kennissystemen die toen werden ontwikkeld, werden echter niet gebruikt in een operationele omgeving, maar waren slechts prototypes. Een van de redenen voor het optreden van dit verschijnsel is dat het onduidelijk is, in welke situaties een kennissysteem een oplossing kan zijn voor een probleem en welke toegevoegde waarde deze oplossing heeft ten opzichte van alternatieve oplossingen. Deze reden vormt de probleemstelling van deze studie.

Om de probleemstelling aan te kunnen pakken wordt eerst een definitie gegeven van een kennissysteem. De in deze studie ontwikkelde definitie is een formeel model, dat de architectuur van een kennissysteem eenduidig beschrijft. Het ontwikkelen van zo'n model was nodig, omdat de in de literatuur gegeven definities te vaag waren om als basis te dienen voor het onderkennen van problemen, die bij voorkeur met behulp van kennissystemen moeten worden opgelost.

De architectuur van een kennissysteem kan worden beschreven met behulp van de volgende componenten:

- Een database-deel, bestaande uit een definitie van de verzameling toegestane databases, een toegelaten database en een controlemechanisme
- Een regelbank
- Een redeneerdeel, bestaande uit een externe regel, een verzameling redeneerregels, een hypothese-selectiefunctie, een regel-selectiefunctie en een redeneerprocedure

Ieder van deze componenten kan afzonderlijk worden benaderd en veranderd, zonder dat rekening hoeft te worden gehouden met de andere componenten. Met behulp van deze componenten is een kennissysteem in staat vragen te beantwoorden door het afleiden van gegevens vanuit de gegeven database met behulp van regels uit de regelbank.

Het architectuurmodel van een kennissysteem kan worden gebruikt voor verschillende doeleinden. Door ieder van de componenten te specificeren ontstaat een conceptuele specificatie van een kennissysteem. Daarnaast levert het model een raamwerk voor het onderling vergelijken van implementatie-tools voor kennissystemen. Door de conceptuele specificatie, opgesteld in termen van het model, te leggen naast de vergelijking van de shells

(in dezelfde termen opgesteld) kan een indruk worden verkregen van de geschiktheid van een specifieke tool voor de implementatie van een kennissysteem.

Om te kunnen herkennen in welke situaties kennissystemen een toegevoegde waarde kunnen hebben boven alternatieve oplossingen dient een systeemontwikkelingsproces een kennistechnologische invalshoek te hebben. Hieronder wordt verstaan dat, behalve het beschrijven van processen en gegevens zoals in traditionele methoden wordt voorgeschreven, ook de kennis expliciet dient te worden beschreven in de vorm van een model van de kennis. Probleemtypen, waar deze aanpak voordelen kan bieden, zijn onder meer diagnoseproblemen en schedulingproblemen. Vaak worden deze problemen niet opgelost volgens een vast algoritme en is de kennis omtrent de oplossing van tevoren niet geformaliseerd (bijvoorbeeld omdat ze slechts aanwezig is in de hoofden van een of enkele personen).

Problemen die een kennistechnologische blik van systeemontwikkeling vragen, hoeven niet per se geschikt te zijn om met behulp van een kennissysteem te worden opgelost. Door zijn specifieke architectuur is een kennissysteem geschikt voor die situaties, waarin bijzondere eisen worden gesteld aan het geven van uitleg over gevolgde redeneringen. Daarnaast is een kennissysteem geschikt om als oplossing te dienen voor situaties, waarin behoefte bestaat aan regelbeheer. Zo'n situatie wordt gekenmerkt door de volgende karakteristieken:

- De regels in de regelbank of het redeneerdeel veranderen regelmatig
- De verzameling regels is groot en/of complex of de redeneerprocedure is ingewikkeld
- De regelbank wordt door meerdere gebruikers tegelijk gebruikt.

Deze karakteristieken zijn afgeleid vanuit ontwikkelingen op het gebied van databases, die hebben geleid tot het ontstaan van DBMS'en. Door een probleemsituatie te beschrijven in termen van de vereiste onderhoudsinspanning kunnen vier situaties worden onderscheiden:

1. Noch voor de gegevens noch voor de kennis bestaat behoefte aan een extra onderhoudsinspanning
2. Alleen voor de gegevens bestaat een behoefte aan een extra onderhoudsinspanning
3. Alleen voor de kennis bestaat een behoefte aan een extra onderhoudsinspanning
4. Zowel voor de gegevens als voor de kennis bestaat een behoefte aan een extra onderhoudsinspanning

In situatie 3 komen kennissystemen in aanmerking als oplossing. Situatie 4 vraagt daarnaast een functionaliteit vergelijkbaar met een DBMS. Deze functionaliteit moet worden geboden door een Knowledge Base Management System.

Kennissystemen moeten om geaccepteerd en werkelijk operationeel gebruikt te worden, geïntegreerd worden in een systeemarchitectuur. Deze integratie kan vanuit twee situaties ontstaan:

- Het kennissysteem wordt stand-alone ontwikkeld. Na ontwikkeling wordt het systeem geïntegreerd met andere informatiesystemen.
- Het kennissysteem vormt een module in een informatiesysteem en wordt ontwikkeld in samenhang met de rest van het systeem.

In beide gevallen zijn de in dit boek ontwikkelde richtlijnen voor het herkennen van geschikte probleemgebieden toepasbaar. Integratie is met name nodig, indien daarmee meervoudige invoer van dezelfde gegevens kan worden vermeden. De keuze voor integratie van kennissystemen in een systeem-architectuur zou ook vanuit een strategische visie moeten gebeuren. In deze visie wordt een kennissysteem gebruikt als middel voor het vastleggen van een "core" van bedrijfsregels. Dit heeft onder meer als effect dat een grotere klantgerichtheid kan worden bereikt. Voor veel bedrijven, opererend in een buyers market, is dit een belangrijke voorwaarde om te kunnen overleven.

SUMMARY

The first knowledge base systems were created in the seventies. Development took mainly place within a research environment. In those years the "classical" knowledge base systems such as XCON, MYCIN, and PROSPECTOR originated. In the eighties business society gained an interest in knowledge base systems. Still, many of the systems that were developed then were never actually used. Most did not pass the prototype stage. One of the causes of this phenomenon is that it is not clear in which situations a knowledge base system is a proper solution. Also, the added value that can be provided by using a knowledge base system compared to alternative solving methods has not been determined satisfactory. This thesis is aimed at providing answers to these two questions.

To answer the questions, a definition of what precisely is meant by a knowledge base system is developed. It consists of a formal model that unambiguously describes the architecture of a knowledge base system. Developing this model was necessary, because a literature survey showed that the definitions found in literature were too vague to act as a basis for recognizing suited problems areas in which knowledge base systems can be used profitably.

The architecture of a knowledge base system can be seen to consist of the following components:

- a database-component, containing a definition of the feasible databases, a feasible database, and a control mechanism,
- a rulebase,
- a reasoning-component, containing an external rule, a set of reasoning rules, a hypothesis-selection function, a rule-selection function and a reasoning procedure

Using these components a knowledge base system can answer a question by deducing the answer from the data provided in the database by using the rules from the rulebase.

The architectural model of a knowledge base system can be used for different purposes. A possible use is to develop the specifications of a knowledge base system by specifying each of the components of the model. Another possibility is to use the model to provide a framework for the comparison of tools to be used for the implementation of a knowledge base system. When the conceptual specification, formulated in the language of the formal model, is compared with a description of a tool noted in the same language, one can judge the suitability of that tool for the implementation of the system.

In this thesis the model is mainly used for the identification of situations in which knowledge base systems have an added value as a solution when compared to other solutions. For this it is required that the system development process includes a knowledge technology perspective. This means that, besides describing the data and processes involved, as required by existing methods, extra attention should be aimed at giving an exact description of the knowledge involved. In addition to data models and process models this results in a knowledge model. Types of problems where this view can be helpful are, among others, diagnosis and scheduling problems. This kind of problem can often not be solved with a fixed algorithm and the knowledge needed for the solution is not formalized beforehand (often, because this knowledge only exists in the brains of some people). An explicit description of the knowledge involved, as provided by a knowledge technological perspective, can aid in clarifying this type of problem.

That problems require a knowledge technology perspective on system development does not necessarily mean that they are suited to be solved using a knowledge base system. A knowledge base system can, due to its specific architecture, serve as a solution when a demand exists for the system clarifying its way of reasoning to the user. Furthermore, a knowledge base system is suited in those situations, where an explicit need for knowledge management exists. Such situations can be characterized by:

- frequent changes in the rulebase or in the reasoning component and/or
- a large and/or complex set of rules and/or
- a complicated reasoning procedure or and/or
- a rulebase that is to be used by several people at the same time.

These characteristics are comparable with earlier characteristics of data stores, leading to the creation of DBMS's. They are all in the area of maintenance. When describing a problem situation from a maintenance point of view, four situations can be distinguished:

1. Neither for the data, nor for the knowledge (rules and/or reasoning procedure) there is a heavy need for maintenance.
2. Only for the data there is a heavy need for maintenance.
3. Only for the knowledge there is a heavy need for maintenance.
4. Both for data and for knowledge is there a heavy need for maintenance.

In situation three knowledge base systems can be considered to be a good solution. Situation four also requires the functionality of a DBMS. This extra functionality should be offered by a Knowledge Base Management System.

To be accepted and used in an operational environment, knowledge base systems have to be integrated within a total system architecture. This kind of integration can be achieved in two ways:

- The knowledge base system is developed stand-alone. After the development, the system is integrated with other information systems.
- The knowledge base system is a module of a wider information system and is developed with the rest of this system.

In both cases the guidelines for identifying suitable problem areas can be used. Integration is essential when it circumvents the need for double input of data. The choice for integrating a knowledge base system into the system architecture should spring also from a more strategic point of view. In this vision a knowledge base system can be used as a means to record a core of business rules of a company. Among others, this will have as an effect that the company can serve its customers in a better way. For many companies this is essential for survival, as they are active in a buyer orientated market.

LITERATUUR

Aerts A.T.M., Alblas G., Hee K.M. van (1991). *Conceptueel modelleren van informatiesystemen.* Academic Services, Schoonhoven.

AI & AA projectgroep (1992). *Selectiesysteem KASSA.* Belastingdienst, Projectgroep AI&AA, Utrecht.

Akkermans J.M., Wielinga B.J., Schreiber A.Th., Balder J.R. (1989). *Naar een formele specificatie van kennismodellen.* Proceedings NAIC-89, Academic Services, Schoonhoven.

Altenkrüger D.E. (1990). *KBMS: aspects, theory and implementation.* Information Systems, Vol. 15, nr. 1.

ANSI/IEEE (1984). *Software Engineering Standards.* The Institute of Electrical and Electronics Engineerings, Inc., New York.

Barker V.E., O'Connor D.E. (1989). *Expert Systems for Configuration at Digital: XCON and Beyond.* Communications of the ACM, vol. 32, nr. 3.

Beckers W.P.A., De Vries Robbé P.F., Zwaard A.M., Mokkink H.G.A., Grol R.P.T.M. (1990). *RAP in ROIS.* Uit (Kwee en van den Herik, 1990).

Beckers W.P.A., De Vries Robbé P.F., Zwaard A.M., Mokkink H.G.A. (1991). *Evaluatie van een programma voor kwaliteitsbewaking in de huisartsgeneeskunde.* Uit (Spek en Treur, 1991).

Beckman T.J. (1991). *Selecting Expert-system Applications.* AI Expert, february

Bemelmans T.M.A. (1991). *Bestuurlijke informatiesystemen en automatisering.* Kluwer Bedrijfswetenschappen/Stenfert Kroese, Leiden.

Boehm B.W. et al (1978). *Characteristics of software quality.* TRW Series of Software Technology, Vol. 1, North Holland Publishing Company, Amsterdam.

Breuker, J.A. et al (1987). *Model-Driven Knowledge Acquisition: Interpretation Models.* Deliverable task A1, Esprit Project 1098, Amsterdam.

Breuker J.A., Wielinga B. (1991). *KADS: A Modelling Approach to Knowledge Engineering.* Universiteit van Amsterdam, Amsterdam.

Brock E.O. de (1990). *De grondslagen van semantische databases.* Academic Service, Schoonhoven.

Broek G.M.C.M. van de, Abbink Spaink H., Schuwer R. (1990). *Ervaringen met specificaties voor kennissystemen.* Informatie, jrg. 32, nr. 6.

Brumsen H., Oostveen A. van, Treur J. (1990). *Specificatie van Modulaire Kennissystemen.* Proceedings NAIC '90, Kerkrade.

Buchanan B.G., Shortliffe E.H. (1984). *Rule-Based Expert Systems (The MYCIN Experiments of the Stanford Heuristic Programming Project).* Addison-Wesley Publishing Company, Reading, Massachusetts.

Chorafas D. (1987). *Applying Expert Systems in Business.* McGraw-Hill, New York.

Clocksin W.F., Mellish C.S. (1987). *Prolog, beschrijving van de standaard.* Kluwer Technische Boeken, Deventer.

Davis R., Buchanan B.G. (1977). *Meta-Level Knowledge: Overview and Applications.* Proceedings IJCAI-77, Cambridge MA.

Davis G.B., Olson M.H. (1984). *Management Information Systems.* McGraw-Hill, New York.

Dumoulin W.J.E. (1989). *Case bureau-eilanden.* Interne notitie, TUE/BDK/BISA, Eindhoven.

Eiben A., Schuwer R. (1991). *Knowledge Base systems, a formal model.* Computing Science Note 91/20, Eindhoven University of Technology.

Erens F. (1990). *Het ontwerp van een produktstruktuur, coderingswijze, documentatiesysteem en configuratiesysteem voor Philips Medical Systems.* Afstudeerverslag TUE, Faculteit Wiskunde en Informatica, sectie Informatica, Eindhoven.

Euwe M., Schuwer R. (1992). *Verkoop Ondersteunend Systeem (V.O.S.).* Doelmatige Bedrijfsvoering, jrg. 4, nr. 6.

Everest G.C. (1986). *Database Management.* McGraw-Hill Book Company, New York.

Frost R. (1987). *Introduction to Knowledge Base Systems.* Collins, London.

Genuchten M. van (1991). *Towards a software factory.* Proefschrift Technische Universiteit Eindhoven.

Harmon P., Maus R. (1988). *Expert Systems Tools and Applications.* Wiley & Sons, New York.

Hayes-Roth F., Waterman D., Lenat D. (1983). *Building Expert Systems.* Addison-Wesley, Reading, Mass.

Heijne den Bak R., Curwiel P. (1989). *Integration of information technology and knowledge technology.* Journal of Software Research, Utrecht.

Herwijnen J. van, Houten E.G. van, Houtsma M.A.W., Romkema H.M. (1990). *Implementatie van een regel-gebaseerd kennissysteem in een relationele database-omgeving.* Informatie, jrg. 32, nr. 1.

Hoog R. de (1992). *KADS-II: de stand van zaken.* In (Hoog en Spek, 1992a).

Hoog R. de, Spek B.R. van der (red.) (1992a). *Kennistechnologie'92. Methoden, technieken en gereedschappen.* Den Haag.

Hoog R. de, Spek B.R. van der (red.) (1992b). *Kennistechnologie'92. Operationele toepassingen, strategische visies.* Den Haag.

Jackson, P. (1987). *Introduction to Expert Systems.* Addison-Wesley, Reading.

Jain H., Chaturvedi A. (1989). *Expert System Problem Selection: A Domain Characteristics Approach.* Information & Management 17, pp. 245-253.

Jong, L.S. de (1988). *Engineering of Expert Systems.* Information and software technology, jrg. 30, nr. 7.

Kolman T., van Nispen A.R.M., Zuijdgeest L. (red.) (1992). *Selectiecriteria voor toepassing van kennistechnologie.* CIAD, Zoetermeer.

Kraats E.J. van de (1989). *Knowledge-based systems: expertise by proxy.* Journal A, vol. 30, nr. 4.

Kuiper A. (1988). *ESDM, een methode voor het ontwikkelen van kennissystemen.* Proceedings AIT'88, Amsterdam.

Kusters R.J., Schuwer R.V. (1991). *An Approach to Integrating Knowledge Base Systems.* Uit (Liebowitz, 1991).

Kwee A.Y.L., Herik H.J. van den (red.) (1990). *AI Toepassingen '90.* Stichting Informatica Congressen, Kerkrade.

Lehman M.M., Belady L.A. (eds.) (1985). *Program Evolution.* Academic Press, London.

Levesque H.J. (1984). *The Logic of Incomplete Knowledge Bases.* Uit: Brodie, Mylopoulos, Schmidt (eds): On Conceptual Modelling, Springer, New York.

Levesque H.J. (1989). *Foundations of Functional Knowledge Representation.* Lecture Notes from 3rd ACAI, Neuchatel.

Liebowitz J.(ed.) (1991). *Expert Systems World Congress Proceedings.* Pergamon Press, New York.

Lloyd J.W. (1987). *Foundations of Logic Programming, Second Edition.* Springer Verlag, Berlin.

Madni A.M. (1988). *The Role of Human Factors in Expert Systems Design and Acceptance.* Human Factors, jrg. 30, nr. 4.

Mars N.J.I. (1988). *Onderzoek van niveau: Kennistechnologie in wording.* Informatie, jrg. 30, nr. 2.

Mars N.J.I. (1991). *Inleiding kennistechnologie.* Academic Services, Schoonhoven.

Martin A., Law R.K.H. (1988). *Expert system for selecting expert system shells.* Information and software technology, jrg. 30, nr. 10.

Mastrigt B. van, Lippolt B.J., Velde C.J. van de (1989). *Slagen en falen van expertsystemen.* Stam Tijdschriften, Rijswijk.

Neches R. (1992). *Sharing and Reuse of Knowledge.* Uit: Mars N.J.I. (ed.), Knowledge Sharing and Reuse: Ways and Means. Workshop Notes ECAI '92, Wenen.

Paulussen R.M.C. et al (1992). *Software-kwaliteit bespreekbaar maken.* Informatie, jrg. 34, nr. 3.

Pels H.J. (1988). *Geïntegreerde informatiebanken.* Stenfert Kroese, Leiden.

Prerau D.S. (1990). *Developing and Managing Expert Systems.* Addison-Wesley Publishing Company, Reading, Massachusetts.

Savory S.E. (1987). *Expertensysteme: Nutzen für Ihr Unternehmen.* Oldenbourg Verlag, München.

Schmidt A.H.J., Oskamp E.W. (1991). *Van KADS naar DESIRE en terug.* Proceedings NAIC'91, Amsterdam.

Schuwer R.V., Kusters R.J. (1990). *Toegevoegde waarde van kennissystemen.* Informatie, jrg. 32, nr. 12.

Schuwer R.V., Pels H.J. (1991). *A unified conceptual view on data- and knowledge bases.* Interne notitie TUE/TBDK/I&T.

Spek B.R. van der, Treur J. (red.) (1991). *Proceedings AIT'91.* Stichting Informatica Congressen, Amsterdam.

Szolovits P. (1989). *Knowledge-Based Systems: A Survey.* Uit: Schmidt J.W. & Thanos C. (eds): Foundations of Knowledge Base Management, Springer, Berlin.

Waterman D.A. (1986). *A guide to expert systems.* Addison-Wesley Publishing Company, Reading, Massachusetts.

Weiss S., Kulikowski C. (1984). *A Practical Guide to Designing Expert Systems.* Rowman & Allanheld, Totowa NJ.

Weitz R., Meyer A. de (1990). *Managing Expert Systems.* Information and Management 19, pp. 115-131.

Wognum P.M. (1990). *Explanation of automated reasoning: How and Why?* Proefschrift, Universiteit Twente, Enschede.

Wognum P.M., Lippolt B.J. (1991). *Wat bepaalt het succes van een kennissysteem?* Informatie, jrg. 33, nr. 10.

INDEX

CURRICULUM VITAE

Robert Victor Schuwer werd geboren op 22 november 1955 te Den Haag. In 1974 behaalde hij het Gymnasium-ß diploma aan het Dominicus College te Nijmegen. Aansluitend heeft hij wiskunde gestudeerd aan de Katholieke Universiteit te Nijmegen. Hier studeerde hij in mei 1980 af op een onderwerp uit de getaltheorie (De vergelijking van Pell). In 1983 is hij als werkstudent begonnen met een studie Informatica aan de (tegenwoordige) Technische Universiteit Eindhoven. Deze studie rondde hij in oktober 1989 af met een onderzoek op het gebied van kennissystemen.

Vanaf augustus 1979 werkte hij part-time als docent wiskunde aan de Rijksmiddelbare Tuinbouwschool te Nijmegen. Vanaf augustus 1980 was hij als docent wiskunde verbonden aan de toenmalige Rijksscholengemeenschap "Den Hulster" te Venlo. In 1984 trad hij in dienst bij Océ-van der Grinten als consultant op het Informatiecentrum. Vanaf oktober 1987 is hij als universitair docent verbonden aan de Technische Universiteit te Eindhoven bij de Faculteit Technische Bedrijfskunde, vakgroep Informatie en Technologie. Hier werd ook het promotie-onderzoek uitgevoerd, dat heeft geleid tot dit proefschrift.

STELLINGEN

behorende bij het proefschrift

Het nut van kennissystemen

van

Robert V. Schuwer

Eindhoven, 12 mei 1993

I

De architectuur is de meest bepalende karakteristiek van een kennissysteem. Zo'n systeem bevat duidelijk onderscheidbare componenten, te weten een database, een regelbank en een redeneermechanisme.
(Bron: dit proefschrift, hoofdstuk 2 en bijlage A.)

II

Zoals is gebleken bij traditionele automatisering geldt ook voor kennissystemen dat een goede conceptuele specificatie vooraf dient te gaan aan een implementatie.
(Bron: dit proefschrift, hoofdstuk 3.)

III

Een goed raamwerk voor het maken van een conceptuele specificatie van een kennissysteem wordt geboden door het XYFAR-model.
(Bron: dit proefschrift, hoofdstuk 3.)

IV

Een belangrijke toegevoegde waarde van een kennissysteem ligt in de mogelijkheid van kennisbeheer binnen organisaties.
(Bron: dit proefschrift, hoofdstuk 5.)

V

Een wezenlijke succesfactor voor kennissystemen is de integratie met anderssoortige systemen.
(Bron: dit proefschrift, hoofdstuk 6.)

VI

Bij het opstellen van architectuurplannen voor toepassingen, zoals Gemeentelijke Functionele Ontwerpen, is een kennistechnologische invalshoek zinvol.
(Bron: Schuwer R.V., Corstens H.J.M. (1992). GFO: van woordenboek naar kennisbank. NGT Geodesia, jrg. 34, nr. 4)

VII

Een afstudeerproject afgestemd op lopend wetenschappelijk onderzoek levert een win-win situatie op voor de afstudeerder, de organisatie waar het project wordt uitgevoerd en de begeleidende wetenschapper.

VIII

Vaak wordt beweerd dat computergebruik zonder technische computerkennis mogelijk moet zijn naar analogie van autogebruik zonder technische autokennis. Deze bewering gaat niet op als onder computergebruik ook het programmeren wordt verstaan.

IX

Het mislukken van veel End User Computing-applicaties wordt veroorzaakt door klakkeloos aan te nemen dat de analogie uit stelling VIII niet mank gaat.

X

Degenen die in de universitaire wereld spreken over onderwijs*last*, kennen hoogstwaarschijnlijk niet het lesgeven aan een heterogene brugklas.

XI

De ernst van een gebeurtenis is evenredig met het aantal "sick jokes" daarover en de snelheid waarmee die ontstaan.

XII

De bewering dat een beoefenaar van wedstrijdsport zich jong blijft voelen, staat op gespannen voet met het feit dat medesporters diezelfde sporter rondom zijn 30e verjaardag al "ouwe" noemen.