Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Kijk op www.nioc2025.nl voor meer informatie.

Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Een intelligent tutorsysteem voor het leren programmeren

E.A. Reinders-Machowska
Vleermuis Software Research
Postbus 2584
3500 GN UTRECHT

Samenvatting

Binnen Vleermuis Software Research is gestart met de realisatie van een Intelligent Tutoring System (ITS) dat gebaseerd is op Volmac Structured Program design (VSP). De VSP-methodiek ondersteunt het ontwerpen van programma's. Dit paper behandelt de wijze waarop het ITS wordt ontwikkeld.

1 Inleiding

Binnen Vleermuis Software Research wordt een intelligent tutorsysteem (ITS) ontwikkeld voor een tamelijk complexe taak: het leren ontwerpen van programma's. In deze beschrijving wordt ingegaan op twee belangrijke vragen. De eerste vraag is wat voor problemen een beginnend programmeur tijdens het leren programmeren ondervindt en wat de oorzaken en oplossingen hiervoor zijn. Om deze vraag te kunnen beantwoorden werd een kort literatuur-onderzoek verricht dat ons een overzicht leverde van de mogelijke problemen van een beginnend programmeur.
De conclusie van dit onderzoek was dat een gestructureerde methodiek voor het ontwerpen van programma's behulpzaam is bij het oplossen van deze problemen. Daarom werd besloten een tutorsysteem te bouwen dat studenten op een gestructureerde wijze leert programmeren. Er werd gekozen voor het leren programmeren volgens de VSP-methodiek. Verder werd direct gekeken of de VSP-methodiek geschikt is om opgenomen te worden in een tutorsysteem. Een domein dat in een kennissysteem opgenomen wordt (en een ITS is een kennissysteem) moet goed gedefinieerd en beschreven zijn en het aantal 'gevoelsmatige' beslissingen tijdens het oplosproces minimaal.

De tweede vraag betreft het ontwikkelproces van het systeem: is het mogelijk om m.b.v. de door ons gekozen methodiek voor het bouwen van kennissystemen, Structured Knowledge Engineering (SKE), zo'n tutorsysteem te ontwikkelen gezien de complexiteit van het domein. SKE (zie Bolesian 1988) is een methodiek die verschillende fasen in de ontwikkeling van kennissystemen onderscheidt, analoog aan de fasering binnen methodieken voor het ontwikkelen van conventionele informatiesystemen zoals SDM en PRODOSTA (zie Smulders 1989).
In dit kader hebben wij behalve de geschiktheid van de SKE-methodiek ook onderzocht of een modulaire opbouw van het systeem de ontwikkeling ervan bevordert en of het mogelijk is een systeem te ontwikkelen dat onafhankelijk is van de programmeertaal die aangeleerd moet worden.

De uit de literatuur bekende ITS'en worden hoofdzakelijk in drie of vier modulen (vaak experts genoemd) verdeeld. Wij hebben voor de volgende indeling van het tutorsysteem gekozen:
- een domeinexpert waarin de kennis van het domein wordt opgeslagen;
- een studentexpert waarin de informatie over de kennis van de cursist en de handelingen die hij uitvoert wordt verzameld;
- een diagnostisch expert die elke stap van de cursist (afkomstig uit de studentexpert) en van de expert (afkomstig uit de domeinexpert) met elkaar vergelijkt en van de eventueel gemaakte fouten vaststelt welke misconcepties hiervan de oorzaak zijn.
- een didactisch expert (coach) die de kennis bevat over toepassingen van didactische strategieën en feed-back geeft aan de cursist.

Allereerst (hoofdstuk 2) wordt een globaal overzicht gegeven van problemen die beginners hebben met het ontwerpen van programma's. Vervolgens (hoofdstuk 3) zal de VSP-methodiek in het kort beschreven worden. In hoofdstuk 4 zal aandacht aan de SKE-methodiek worden besteed en in hoofdstuk 5 volgt een beschrijving van de vier bovengenoemde modulen (experts). In het laatste hoofdstuk (6) worden de conclusies en de mogelijke toekomstige activiteiten vermeld.

2 Problemen met programmeren

Uit literatuuronderzoek is gebleken dat een beginnend programmeur altijd met een aantal problemen worstelt. Uit onderzoek naar de verschillen tussen ervaren en beginnende programmeurs (verder respectievelijk ook

experts en cursisten of beginners genoemd) is gebleken dat de beginners grote problemen hebben met de aanpak van een programmeerprobleem, het opstellen van algoritmen en met een foutloze implementatie ervan. De oorzaken van deze problemen zijn verschillend.
De oorzaak van een verkeerde aanpak van het probleem ligt onder andere in het feit dat cursisten niet goed begrijpen wat het programma voor ze kan doen en hoe het op een computer uitgevoerd wordt. Ze nemen nauwelijks de tijd om goed inzicht in het probleem te krijgen en beginnen vaak de oplossing direct in de programmeertaal te schrijven. Zelden verdelen ze een probleem in subproblemen en als ze dat wel doen houden ze meestal geen rekening met randvoorwaarden en speciale gevallen. Ook definiëren ze dikwijls één begrip op meerdere wijzen, waardoor fouten ontstaan.
Tijdens het ontwerpen van algoritmen treden beginners direct in de kleinste details met als gevolg dat ze de hoofdlijn uit het oog verliezen. Eén van de belangrijkste oorzaken van problemen met implementatie is het feit dat beginners het omzetten van constructies in hun natuurlijke taal (bijvoorbeeld Nederlands) naar programmacode moeilijk vinden. Ze maken vaak geen onderscheid tussen de betekenis van woorden en constructies uit een natuurlijke taal en een programmeertaal. Ze kijken alleen naar oppervlakte syntaxis en niet naar de onderliggende betekenis van natuurlijke talen en programmeertalen.
Om de cursist zo snel mogelijk te leren programmeren moet hij tijdens het leerproces zodanig steun krijgen dat hij gedwongen wordt om over bovengenoemde zaken goed na te denken.

Tenslotte zijn er grote verschillen tussen experts en beginners bij het verbeteren van fouten in al geimplementeerde programma's. Van de verschillende soorten fouten die het programma kan bevatten zijn de semantische en logische fouten het moeilijkst te verbeteren. Experts besteden de meeste tijd aan het opsporen van dit soort fouten. Beginners komen daar niet aan toe want vaak merken ze zelfs niet dat het programma uiteindelijk iets anders doet dan wat men ervan verwachtte.

3 Beschrijving van VSP

De meeste programmeercursussen (en ook tutorsystemen) zijn hoofdzakelijk gericht op syntaxis. Indien men de syntactische constructies van een programmeertaal goed kent betekent dit echter niet dat men goed kan programmeren. Uit verschillende onderzoeken blijkt dat het op een gestructureerde wijze ontwerpen van

programma's een oplossing biedt voor veel problemen van de beginnend programmeur. Een bekende methodiek voor gestructureerd programmeren is JSP (Jackson Structured Programming).
Binnen VOLMAC Nederland B.V. is deze methodiek aangepast aan de interne behoeften en standaards van het bedrijf en wordt ze VSP (Volmac Structured Program design) genoemd. In principe moet elke medewerker van VOLMAC Nederland B.V. volgens deze methodiek programma's leren ontwerpen. De VSP-methodiek biedt een soort algoritme voor het bouwen van programma's. Door dit algoritme te hanteren leert de cursist het probleem gestructureerd te analyseren en naar formele structuren te vertalen. Evenals JSP gaat VSP uit van het principe, dat de structuur van een programma identiek is aan een logische combinatie van de structuren van bestanden die door het programma benaderd worden.
Gezien het feit dat het door ons gebouwde tutorsysteem in eerste instantie voor het coachen van cursussen binnen VOLMAC Nederland B.V. ingezet zou moeten worden hebben wij voor het opnemen van de VSP-methodiek in het ITS gekozen.

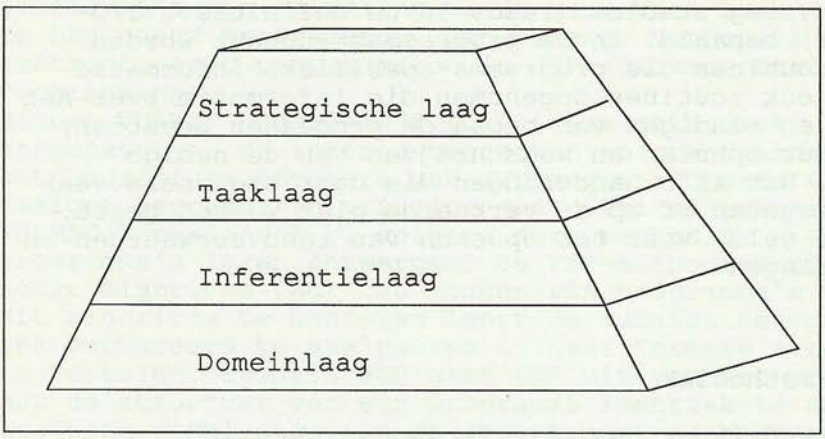
Ten behoeve van het tutorsysteem hebben wij het ontwerpen van programma's volgens de VSP-methodiek in vijf fasen verdeeld:
- formalisering van de programmabeschrijving (d.w.z. van het technisch ontwerp);
- het maken van een definitieve I/O-structuur (deze structuur representeert de relaties tussen de records van de verschillende bestanden die door het programma worden verwerkt);
- het maken van een programmastructuur en een overzicht van acties die binnen elke routine (procedure) plaatsvinden;
- optimalisatie van de programmastructuur en het coderen van de hoofdlijn van het programma;
- detaillerings- en coderingsfase (detailleren van routines m.b.v. detailleringstechnieken, codering van routines en compilatie van het programma);

Men kan zich natuurlijk afvragen hoe zo'n gestructureerde methodiek als VSP de beginnende programmeur kan helpen bij het oplossen van zijn problemen. De activiteiten die de programmeur in eerste instantie moet ondernemen hebben onder andere tot doel om de relaties te leggen tussen de invoer en de uitvoer van het programma. Hierdoor wordt de programmeur van het begin af aan gedwongen het probleem goed te analyseren, erover na te denken en het te specificeren, d.w.z. het gefaseerd en gestructureerd aan te pakken. Zo worden de plaats en de volgorde waarin de procedures binnen het programma uitgevoerd moeten worden, al in

een vrij vroeg stadium (reeds in de definitieve I/O-structuur) bepaald. In de programmastructuur worden behalve routines die programma-specifieke informatie bevatten ook routines opgenomen die informatie over het beginnen en eindigen van bepaalde processen bevatten, alsmede het ophalen en wegschrijven van de nodige gegevens. Dit zijn handelingen die door beginners vaak worden vergeten of op de verkeerde plek worden gezet. Hetzelfde geldt voor het opnemen van randvoorwaarden en uitzonderingen.

4 De SKE-methodiek

Zoals gezegd is er besloten om het project uit te voeren volgens de SKE-methodiek. Om een kennissysteem te kunnen bouwen moet eerst de hele kennis waarvan het systeem gebruik maakt (zgn. domeinkennis) in kaart gebracht worden. De kern van de SKE-methodiek, de kennisanalyse wordt ondersteund door een aantal wetenschappelijke methodologieën, waarvan de KADS-methodologie (zie Breuker e.a. 1987) de belangrijkste is. Bij de analyse van expertkennis maakt de SKE-methodiek gebruik van modellen van probleemoplosprocessen.
Binnen de SKE-methodiek kan men vier fasen onderscheiden. Eerst wordt een initiatiestudie uitgevoerd waarin een onderzoek wordt gedaan naar de expertactiviteiten. Vervolgens wordt de kennisverwervingsfase uitgevoerd. In deze fase wordt de expertkennis verzameld en geanalyseerd en op grond daarvan wordt een conceptueel kennismodel opgesteld. In de derde fase wordt het functioneel en technisch ontwerp van het hele systeem uitgevoerd. In de laatste (vierde) fase vindt de implementatie van het systeem plaats.
De belangrijkste fase van het ontwikkelingstraject volgens de SKE-methodiek is de kennisanalyse. Het doel van de kennisanalyse is de beschrijving van de kennis van de expert in vier lagen (zie figuur 1). Deze onderverdeling is gebaseerd op de verschillende rollen die de kennis kan spelen in het probleemoplosproces.



Figuur 1
De kennislagen van een interpretatiemodel

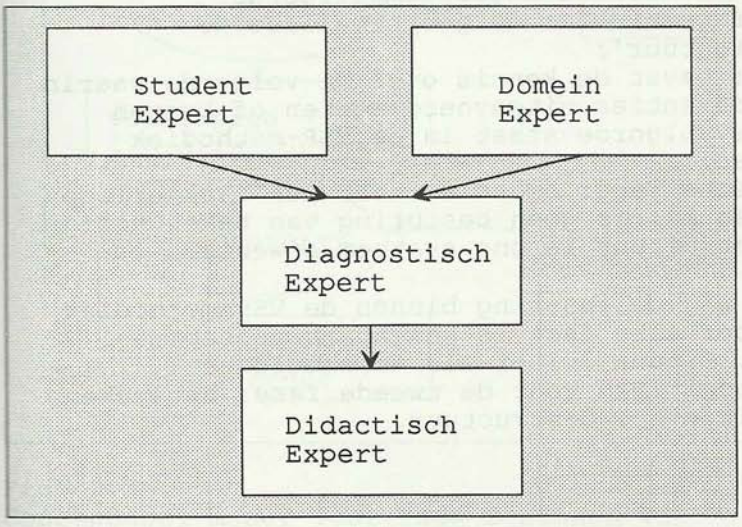
De eerste laag, de zgn. domeinlaag, bevat de statische kennis van het domein. Daarin worden de vakspecifieke begrippen en hun onderliggende relaties vastgelegd. De tweede laag is de zgn. inferentielaag. Daar wordt de kennis met behulp van interpretatiemodellen beschreven. Bij de derde laag, de zgn. taaklaag wordt de volgorde beschreven waarin de inferenties moeten of kunnen worden uitgevoerd. Ook de communicatie met de gebruiker en met externe bestanden wordt hier gedefinieerd. Het laatste niveau is de zgn. strategische laag. Deze laag bevat de kennis die het systeem in staat stelt de volgorde waarin verschillende taken worden uitgevoerd te bepalen en eventueel een beslissing te nemen over het wel of niet uitvoeren van een bepaalde taak. De kennis is hier georganiseerd in een processtructuur die de flexibiliteit van het probleemoplossen op expertniveau weerspiegelt.

Een belangrijk hulpmiddel binnen de SKE-methodiek bij het bouwen van modellen is een bibliotheek met interpretatiemodellen. Zoals gezegd beschrijven de interpretatiemodellen de kennis in de inferentielaag. De interpretatiemodellen worden opgesteld voor de verschillende typen generieke taken. Generieke taken zijn bijvoorbeeld beoordeling, bewaking, verschillende soorten diagnose- en ontwerptaken, en zijn domeinonafhankelijk. In een interpretatiemodel wordt aangegeven welke gevolgtrekkingen wel en niet kunnen worden gemaakt op grond van de kennis die op het domeinniveau aanwezig is. Interpretatiemodellen zijn behulpzaam bij het opstellen van conceptuele modellen. Een conceptueel model is het resultaat van de kennisverwervingsfase binnen de SKE-methodiek en kan gezien worden als een interpretatiemodel dat de

beschrijving van de voor een specifiek systeem benodigde kennis en redeneringen binnen een bepaald domein bevat. Het conceptueel model wordt verder gebruikt voor het ontwerpen en de implementatie van het systeem.

5 Ontwikkeling van het tutorsysteem

Zoals al eerder werd vermeld gaan wij bij de ontwikkeling van het ITS voor VSP uit van een verdeling van het systeem in vier afzonderlijke modulen (experts) (zie figuur 2).



Figuur 2
Overzicht van de experts in het ITS

Hieronder wordt de inhoud en de functionaliteit van elk van deze modulen beschreven.

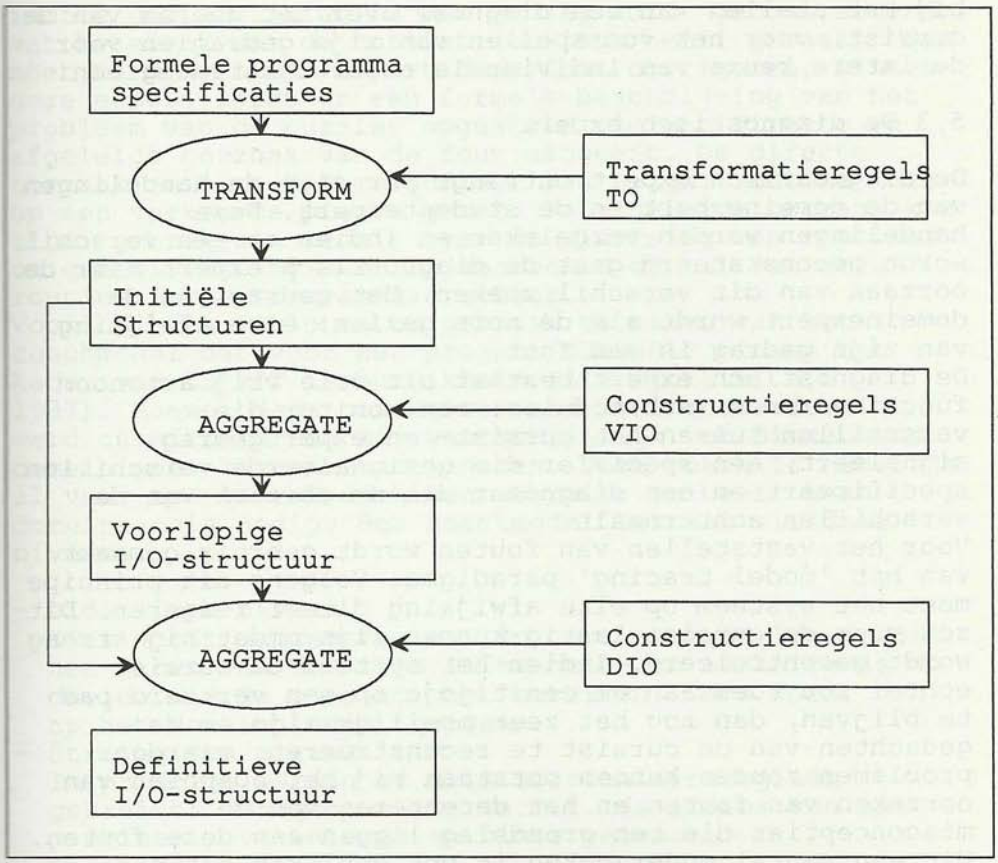
5.1 De domeinexpert

Het probleemoplossend gedrag van een ervaren programmeur bij de ontwikkeling van programma's is vaak gebaseerd op ervaring. De ervaren programmeur kijkt meestal vooruit en neemt al in een vroeg stadium beslissingen die direct tot een zo optimaal mogelijke structuur van het programma leiden: hij maakt gebruik van heuristieken. Men kan dit soort gedrag niet van de beginnende programmeur verwachten. Tijdens zijn leerproces moet de beginner gedwongen worden om zeer systematisch te werken, wat betekent dat alle stappen

expliciet uitgevoerd moeten worden. Ons systeem biedt daarom geen plaats voor heuristieken.
Het bouwen van een programma is een ontwerptaak. Voor het ontwerpen van programma's volgens de VSP-methodiek hebben wij een algemeen interpretatiemodel voor ontwerptaken gekozen. Zoals gezegd wordt de kennis van de expert in vier lagen beschreven. Voor VSP is het beeld als volgt:

- de domeinlaag: bevat de hele kennis over het ontwerpen van het programma volgens de VSP-methodiek (volgorde, regels...);
- de inferentielaag: hierin wordt de kennis opgenomen over de mogelijke inferenties op het domeinniveau, zoals 'transformeer de ongeoptimaliseerde programmastructuur naar de geoptimaliseerde programmastructuur';
- de taaklaag: bevat de kennis over de volgorde waarin bepaalde inferenties uitgevoerd moeten of kunnen worden. Deze volgorde staat in de VSP-methodiek tamelijk vast;
- de strategische laag: omdat er maar één taakverloop te definiëren is, is geen besturing van taken nodig en is deze structuur in ons systeem afwezig.

Eerder hebben wij de fasering binnen de VSP-methodiek aangegeven. Voor elke fase is een apart conceptueel model gebouwd. Figuur 3 laat als voorbeeld het conceptueel model zien voor de tweede fase: het maken van een definitieve I/O-structuur.



Figuur 3
Conceptueel model voor fase 2 binnen VSP

5.2 De studentexpert

Deze expert moet de informatie over de kennis en de handelingen van de cursist onthouden. Het doel van het systeem is niet zozeer de cursist verschillende feiten en regels, maar strategieën te leren om programma's zo snel en zo goed mogelijk te kunnen ontwerpen. Daarom wordt in deze module alleen de kennis van de cursist op inferentie- en taakniveau opgenomen. De kennis van feiten, dus de kennis op domeinniveau wordt alleen als een soort documentatie bewaard.
De studentexpert moet onthouden welke fouten de cursist heeft gemaakt en welke stappen hij wel of niet kent. Op deze wijze zou het mogelijk zijn een soort geschiedenis van het gedrag van de cursist te creëren, wat indirect kan leiden tot inzicht in zijn leerproces. Deze informatie (bijvoorbeeld over de behandelde onderwerpen en de ondernomen onderwijsacties) kan gebruikt worden

bij het stellen van een diagnose over het gedrag van de cursist, voor het voorspellen van zijn gedrag en voor de latere keuze van individuele onderwijsstrategieën.

5.3 De diagnostisch expert

De diagnostisch expert ontvangt per stap de handelingen van de domeinexpert en de studentexpert. Deze handelingen worden vergeleken en indien er een verschil wordt geconstateerd gaat de diagnostisch expert naar de oorzaak van dit verschil zoeken. Het gedrag van de domeinexpert wordt als de norm gezien: elke afwijking van zijn gedrag is een fout.
De diagnostisch expert bestaat uit drie vrij autonoom functionerende (sub)modules: een monitor die verschillen tussen het cursist- en expertgedrag signaleert, een specifier die gesignaleerde verschillen specificeert en een diagnoser die de oorzaak van de verschillen achterhaalt.
Voor het vaststellen van fouten wordt gebruik gemaakt van het 'model tracing' paradigma. Volgens dit principe moet het systeem op elke afwijking direct reageren. Dit zou voor de cursist lastig kunnen zijn omdat hij streng wordt gecontroleerd. Indien het systeem de cursist echter zou toestaan om een tijdje op een verkeerd pad te blijven, dan zou het zeer moeilijk zijn om de gedachten van de cursist te reconstrueren, waardoor problemen zouden kunnen ontstaan bij het opsporen van oorzaken van fouten en het detecteren van de misconcepties die ten grondslag liggen aan deze fouten. Uit een aantal onderzoeken is ook gebleken dat een cursist die te lang op het verkeerde pad blijft later problemen heeft om op het correcte pad terug te keren. Een strenge begeleiding van de cursist komt overigens vrijwel overeen met de werkelijke cursussituatie. Tijdens de cursus zoals bij VOLMAC gegeven, moet de cursist elke stap expliciet uitvoeren, met de coach bespreken en dan pas mag hij verder gaan.
Men moet er echter voor zorgen dat het systeem ook niet te star wordt. Dat kan de cursisten demotiveren om met het systeem te werken. Daarom kan de cursist bijvoorbeeld, indien er keuze bestaat uit meerdere handelingen met betrekking tot een bepaald doel, zelf kiezen wat hij wil gaan doen. De domeinexpert zal dan zijn keuze respecteren en navolgen.

5.4 De didactisch expert (coach)

De input voor deze expert is de diagnose die de diagnostisch expert over het gedrag van de cursist heeft gesteld. Indien de stap van de cursist overeenkomt met de stap van de expert dan wordt de cursist geprezen om zijn motivatie hoog te houden.

Indien een afwijking in het gedrag wordt geconstateerd, wat betekent dat de cursist een fout heeft gemaakt, dan moet de didactisch expert deze fout corrigeren. Binnen deze expert wórdt er een formele beschrijving van het probleem van de cursist opgesteld die een directe en afgeleide oorzaak van de fout aangeeft. De directe oorzaak (m.b.t. VSP-methodiek) kan zijn: routine X is op een verkeerde plaats gezet. De afgeleide oorzaak kan zijn: gebrek aan kennis over de plaatsing van routines of een misvatting over bijvoorbeeld de rol van deze routine in de programmastructuur.
Voorlopig is besloten om gebruik te maken van het coachmodel dat voor het project EUROHELP binnen het kader van ESPRIT is ontwikkeld (zie Winkels, e.a. 1987). Hoewel deze coach voor een heel ander domein werd ontwikkeld zou hij waarschijnlijk ook binnen ons domein goed kunnen functioneren. Deze coach is namelijk al voor een aantal domeinen gebruikt en heeft geen domeinkennis nodig. Het coachmodel bestaat uit drie niveaus:

- Didactisch niveau
  Hier wordt aangegeven wat de cursist moet leren. Bij het leren van VSP zijn inferentie- en taakniveaus de doelen die de cursist moet bereiken. Soms wordt ook op het domeinniveau ingegaan.
- Strategisch niveau
  In principe worden er drie soorten strategieën gebruikt: terugkoppeling (m.b.t. de status van het probleem), verbeteren en hints geven.
- Tactisch niveau
  Hier wordt de communicatie tussen het systeem en de gebruiker bepaald. De cursist kan algemene feed-back krijgen (bijvoorbeeld 'dat is goed', of 'denk nog even na') of feed-back met betrekking tot de domeinkennis.

Het op een tactisch niveau bepaalde optreden van de coach wordt in eerste instantie in een formele representatie opgesteld. Naar de gebruiker toe moet het echter in een andere vorm omgezet worden. Dat kan een tekst in natuurlijke taal zijn (een uitleg) of een tekening of iets dergelijks.

6 Afsluiting

Uit het beschreven onderzoek komen een aantal relevante conclusies naar voren. Ten eerste is gebleken dat de VSP-methodiek zich goed leent als domein van een intelligent tutorsysteem. In de bestaande bronnen over deze methodiek zijn weliswaar slechts enkele delen van programmabouw op formele wijze beschreven, maar door

middel van andere kennisverwervingtechnieken als interviews en hardopdenk-sessies waren wij toch in staat de benodigde kennis over alle stappen binnen de VSP-methodiek te verzamelen.
De tweede, belangrijke conclusie is dat SKE een effectieve methodiek blijkt te zijn voor het ontwikkelen van een ITS. Het grootste deel van de domeinexpertise met betrekking tot programmabouw volgens de VSP-methodiek kon met behulp van SKE gemodelleerd worden. De uitgevoerde analyse heeft tot een doorzichtige en eenvoudige domeinexpert geleid.
Het is ook gebleken dat de modulaire opbouw een voordeel kan bieden bij het bouwen van tutorsystemen met een vrij complex domein. Ons systeem is in zijn geheel modulair opgebouwd: het bestaat uit vier hoofdmodulen (experts) die op hun beurt weer uit een aantal kleinere (sub)modulen bestaan waarin weer nieuwe onderdelen onderscheiden kunnen worden. De functioneel verwante kennis wordt binnen één module (of groep modulen) verzameld.
De modulaire opbouw van het systeem maakt het systeem doorzichtig en beheersbaar. De modulen worden onafhankelijk van elkaar ontwikkeld, waardoor het aantal mogelijke inferenties binnen één module beperkt blijft. Men moet uiteraard wel zorgen voor een goede communicatie tussen de modulen.

In het door ons ontwikkelde tutorsysteem moet vaak het uiteindelijke programma in COBOL geschreven worden. De vraag rees of het mogelijk is een programmeertaal-onafhankelijk tutorsysteem te ontwikkelen. Het feit dat de laatste fase van het ontwerptraject, waarin het programma gecodeerd wordt, totaal taalafhankelijk is ligt voor de hand. Bij de analyse bleek echter dat men al in een veel eerder stadium van het programmabouwtraject rekening houdt met de programmeertaal. Meestal wordt dat gedaan om efficiency redenen. De cursist is echter niet in staat om dat te doen. Daarom hebben wij besloten om in de eerste fasen van het ontwikkelingstraject geen rekening met de programmeertaal te houden.
Indien besloten wordt meer programmeertalen in het systeem op te nemen moet eerst een vergelijkend onderzoek tussen verschillende programmeertalen uigevoerd worden. Er zijn aanwijzingen dat het mogelijk moet zijn een taalonafhankelijk tutorsysteen te bouwen. Mocht dit niet het geval zijn dan zullen dankzij de gebruikte methodiek en modulaire opbouw van het systeem de aanpassingen van het systeem alleen plaatselijk zijn en tot het domeinniveau beperkt blijven.
De uitbreiding van het door ons ontwikkelde systeem kan behalve de toevoeging van een aantal programmeertalen nog een andere richting opgaan. Een andere mogelijkheid

is het aanpassen van het systeem voor het coachen van de JSP-methodiek. De VSP-methodiek, waarop het ITS is gebaseerd, vertoont sterke verwantschap met de JSP-methodiek, de meest bekende methodiek voor gestructureerd programmeren. In beide methodieken is de programmastructuur gebaseerd op de structuur van de gegevens die door het programma worden verwerkt en aangemaakt. Ook de handelingen om tot een gestructureerd programma-ontwerp te komen zijn sterk aan elkaar gerelateerd. Uiteraard zijn er ook verschillen tussen beide methodieken, maar deze zijn niet structureel. Dat houdt in dat het mogelijk moet zijn een op de VSP-methodiek gebaseerd ITS om te bouwen tot een systeem dat geschikt is voor het coachen van de JSP-methodiek. Bij het analyseren van de domeinkennis voor de JSP-methodiek zouden veel aspecten uit de VSP-methodiek overgenomen kunnen worden. Zo'n aanpassing is mogelijk dankzij de modulaire opbouw van het kennisdomein in het ITS voor de VSP-methodiek. De domeinexpert is in principe de enige component van het ITS die aangepast zou moeten worden. De andere drie componenten (studentexpert, diagnostisch expert en coach) kunnen zonder veranderingen worden overgenomen.

Aan de ontwikkeling van dit kennissysteem hebben verder Jolanka Eefting en Norbert Dullaart een belangrijke bijdrage geleverd.

Gebruikte literatuur

- tijdschriften:
  Reinders-Machowska, E.A., Eefting, J.E.F., Dullaart, N.A.J. (1989), "Waarom schrijven uw cursisten zulke kromme programma's" en bijlage. Journal of Software Research nr.3, p. 11-23.
  Smulders, J. (1990), Structured Knowledge Engineering; een methodiek voor het ontwikkelen van kennissystemen. Informatie nr 5.
  Winkels, R.G.F., Achthoven, W. & Gennip, A. van, (1989) Methodologie en Modulariteit in ITS-ontwerp, Journal of Software Research nr. 1, p. 15-20.

- boeken/rapporten:
  Bolesian B.V., (1988) Syllabus van de cursus "Structured Knowledge Engineering".
  Breuker, J.A., Wielinga, B., Someren, M.van, Hoog, R.de, Schreiber, G., Greef, P.de, Bredeweg, B., Wielemaker, J. & Billeault, J.,(1987) 'Model-driven Knowledge Acquisition. Interpretation Models'. Rapporttaak A1, ESPRIT Project 1098.
  Jansen, H. (1984) "JSP Jackson Structureel Programmeren". Academic Service, Den Haag.

Volmac TopTraining B.V.(1988) "Handboek Programmabouw".
Winkels, R.G.F. (1987). PSP-A Prototype Strategy Planner for IHSs. Rapport 2.2.4 van het ESPRIT Project 280. 'EUROHELP'. Memo 81 van het VF-Project 'Knowledge Acquisition in Formal Domains'. Universiteit van Amsterdam.
Winkels, R.G.F. & Breuker, J.A. (1988) Een Computer Coach voor het IIV Project. Rapportage van het Intelligent Video System project.