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.

Prototyping van COO m.b.v. een workbench

J.H. Verpoorten
Vakgroep Computer & Letteren
Universiteit Utrecht
Achter de Dom 22
3512 JP Utrecht

Samenvatting

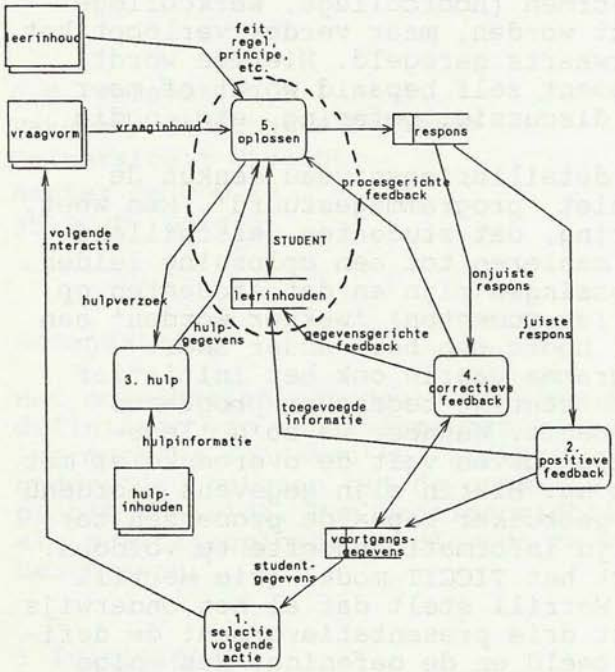
Het ontwikkelen van courseware is identiek aan het definiëren van een leermachine. Gesteld wordt dat de leermachine fundamenteel afwijkt van het 'warme' onderwijs, hetgeen bij de ontwikkeling tot bepaalde problemen leidt. Een modelgeoriënteerde ontwikkeling als praktijkoplossing, alsmede twee voorbeelden, worden beschreven.

1 Inleiding

De term 'computerondersteund onderwijs' is op een bepaalde manier onjuist: niet de computer ondersteunt het onderwijs, maar het programma. In het programma is vastgelegd wat er op bepaalde momenten gebeurt, zodat we beter kunnen spreken van een leermachine. Dat deze machine met behulp van de computer is gerealiseerd, is (in eerste instantie) niet zo belangrijk. Ook bijv. de mechanische Skinnerdisk en de Autotutor (zie Lumsdaine 1960) zijn leermachines, waarin een visie op leren is uitgedrukt, zichtbaar in de wijze van ordenen van de lesstof en de soort activiteiten die de student ondergaat. Deze twee leermachines zijn beiden geheel programma-gestuurd, hetgeen zeggen wil dat de opeenvolging van de lessegmenten en de activiteiten door de machine bepaald worden. Het tijdsaspect (wat gebeurt er op welk moment) speelt hierbij een grote rol, niet alleen op het hoge niveau van de lesonderdelen, maar ook op het lage niveau van afzonderlijke vragen. Dit vereist een veel gedetailleerder kijk op de inrichting van het onderwijs dan de 'gewone' docent gewend is. Maar niet alleen dat. Er ontstaan ook problemen bij het bepalen van bijvoorbeeld een feedback-actie. Het is heel moeilijk om een foutsituatie 1:1 te matchen met één bepaalde foutieve handeling. Eenzelfde fout kan meerdere bestaansgronden hebben, net zoals één verkeerd inzicht tot verschillende fouten kan leiden.
De docent is niet gewend het onderwijs zo gedetailleerd

van te voren in te richten. In de dagelijkse praktijk van het verzorgen van onderwijs wordt wel van te voren bepaald welke onderwerpen op welk moment behandeld worden en welke werkvormen (hoorcollege, werkcollege etc.) daarbij gebruikt worden, maar verder verloopt het onderwijs meer achterwaarts geregeld. Hiermee wordt bedoeld dat op het moment zelf bepaald wordt of meer uitleg, voorbeelden, discussie, oefening etc. nodig is.
Behalve op een hoger detailleringsniveau denken de meeste docenten ook niet 'programmagestuurd'. Men weet, al dan niet uit ervaring, dat studenten verschillend leren, dat vaak meer manieren tot een oplossing leiden, er soms meerdere oplossingen zijn en dat studenten op verschillende wijzen (en momenten) 'wakker worden' aan de oefenstof. Hierbij hoort een heel ander soort 'programma', een programma waarin ook het initiatief van de student (studentsturing tegenover programmasturing) zijn plaats heeft. Wanneer we zo'n 'leermachine' gestalte willen geven valt de overeenkomst met het informatiesysteem op. Hierin zijn gegevens geordend aanwezig en staan de gebruiker bepaalde processen ter beschikking om aan zijn informatiebehoefte te voldoen. In grote trekken werkt het TICCIT-model (zie Merrill 1980) op deze wijze. Merrill stelt dat al het onderwijs kan worden herleid tot drie presentatievormen: de definitie/regel, het voorbeeld en de oefening. Het enige wat de docent doet is de lesstof opdelen in segmenten en daar, op meerdere presentatieniveaus, uitleg, voorbeelden en oefeningen bij verzorgen. Dit gaat in de database, waarna de student zijn eigen instructiepad bepaalt door de keuze van lessegmenten en de activiteiten daarbinnen. Men kan zeggen dat in dit onderwijskundige informatiesysteem door middel van een studentgestuurde opeenvolging van verschillende activiteiten (uitleg, voorbeeld, oefening) op verschillende niveaus een bepaald (en voor die student uniek) leerproces plaats vindt.
Op zich beschouwd eenvoudig te ontwikkelen: een docent hakt zijn leerstof in hapklare brokken -'cooking an elephant is breaking it into smaller pieces ' (zie Motill 1984)- bepaalt werkvormen erbij, programmeert deze (of laat dat doen) en distribueert zijn onderwijs op de een of andere fysieke wijze. Op zich is dit juist, maar het 'oplossysteem' dat 'student' heet hanteert gegevens (definitie/regels e.d., kortom leerinhouden) in samenhang met processen (zoals oplosstrategieën) en zit buiten het informatiesysteem. Hoe bepalen we dan of feedback toereikend is, aan de informatiebehoefte voldoet? Anders gezegd: een bepaalde fout kan voortkomen ofwel uit gebrek aan kennis om het probleem mee op te lossen, of een verkeerd gekozen methode van oplossen; het probleem kan liggen aan de gegevens-

kant of aan de proceskant.
Dit wordt m.b.v. het volgende DFD in beeld gebracht:



Figuur 1
Data Flow Diagram van een docent/student interactie

De docent ontmoet kortom bij het bedenken van een leermachine tal van moeilijkheden. Besluiten ter zake van lesstofordening en studentactiviteiten moeten van te voren worden genomen, waar men gewoon is dit met 'achterwaartse regeling' op te lossen. Uit een programma- of studentgestuurde opzet van de les vloeien consequenties voort m.b.t. feedbackverzorging, doceerstijl en leerstijl. Consequenties welke door de docent zelden zo expliciet tegemoet getreden worden.

2 Het ontwikkeltraject

2.1 De specificatiefasen

In de informaticapraktijk zien we dat softwareprojecten een gefaseerde ontwikkeling kennen, verlopend van specificatie naar realisatie, o.a. om reden van projectbeheersing. In het veel gebruikte SDM (zie Turner e.a. 1986) worden deze fases aangeduid met informatieplanning, definitiestudie, basisontwerp, detailontwerp, realisatie, invoering en gebruik/beheer, waarbij elke

fase en de behorende methoden en technieken nauwkeurig uitgewerkt zijn.
In deze gefaseerde ontwikkeltrajecten is het de bedoeling dat er een geleidelijke transformatie van specificatie naar realisatie plaatsvindt. In de specificatiefasen bevinden we ons op algemeen beschrijvend niveau, om uiteindelijk in de realisatiefase op microniveau uit te komen. In elke fase neemt de detaillering toe doordat het probleem preciezer beschreven wordt (stepwise refinement, topdown functionele decompositie).
Hierbij wordt er impliciet vanuit gegaan dat er voldoende inzicht is op detailniveau m.b.t. feiten (gegevens) en deeloplossingen (processen) om elke volgende stap in het transformatieproces te kunnen zetten. Hier ontmoeten we problemen. Ten eerste hebben de topdown gerichte ontwikkeltechnieken vooral betrekking op de procesdecompositie, terwijl in het onderwijs de gegevens (indelen van lesstof) de eerste aandacht krijgen. De ervaring bij een aantal coo-projecten is dat docenten bij het ontwerp in eerste instantie uitgaan van de structuur van de lesstof en pas daarna de werkvormen erbij bepalen. Gegevens en processen horen echter wel bij elkaar. In elk systeem zijn gegevens zowel invoer als uitvoer van een proces; gegevens welke niet door een proces verwerkt cq geleverd zijn, zijn redundant. In het onderwijs hebben de werkvormen en de lesstof betrekking op de gestelde eindtermen, maar welke docent formuleert eindtermen voor elk onderdeel van zijn les? Ten tweede zijn oplosprocessen niet eenduidig te bepalen, m.a.w. er is meer dan één adequaat leerprogramma, en vindt het oplosproces plaats in het hoofd van de student, dus buiten het systeem. Ten derde is het vaak moeilijk te zeggen waar precies het 'wat' van het specificeren ophoudt en het 'hoe' van de realisatie begonnen is. Het 'wat' en het 'hoe' zijn nauw aan elkaar verwant. Vaak maakt een technische mogelijkheid een bepaalde oplossing zichtbaar. Een docent ziet daarin dan een voor hem unieke oplossing van een probleem. Dit rust op de grondgedachte dat een probleem oplossen hetzelfde is als het probleem zo weergeven dat de oplossing doorzichtig wordt (zie Van Amstel 1987). Hiermee hebben we echter nog geen leermachine.

Niet alleen staan de docent weinig ontwikkeltechnieken ter beschikking, passend bij de specifieke problematiek van het computerondersteund onderwijs, ook worden aan de leermachine andere eisen gesteld dan men in de dagelijkse onderwijspraktijk gewend is. Hoe kan men specificaties opstellen als men aan de basis nog niet weet wat mogelijke oplossingen kunnen zijn? Dit is de 'open deur': men vindt meestal alleen iets als men weet wat men zoekt.

2.2 De realisatiefase

De realisatiefase omvat meer dan het bouwen alleen, maar daartoe beperken we ons even. Er is bij programma's meestal een bepaalde verhouding tussen gebruikersvriendelijkheid en transparantheid. Een menugestuurd programma geeft direct alle mogelijkheden weer en het volstaat er een keuze uit te maken. Bij een commandogestuurd programma moet men zich niet alleen de commando's zelf herinneren, maar daar vaak ook m.b.v. parameters een bepaalde uitwerking aan mee geven. Daardoor blijft er wel vaak meer open voor de gebruiker. Ook bij auteurstalen zien we deze spanning tussen gebruikersvriendelijkheid en transparantheid. Een algemene programmeertaal (C, Pascal e.d.) is behoorlijk transparant, maar voor de docent onvriendelijk. In de uitdrukkingen van de programmeertaal kan hij zijn onderwijs niet terugvinden. Docenten gebruiken dus liever een auteurstaal of -systeem. Wat is nu eigenlijk de winst? Een auteurstaal biedt de auteur een ander venster op het programmeren, zonder het programmeren werkelijk fundamenteel veranderd te hebben. Tegelijk worden bepaalde technische aspecten afgeschermd. Zowel in TAIGA als in Authorware bijvoorbeeld praten we nog steeds over 'variabelen' van een bepaald 'type', en wordt de flow van het programma bepaald door IF<conditie>THEN<statement1>ELSE<statement2>-constructies. De voor de programmeur werkelijk interessante variabelen (omdat daarmee veel te bereiken is) zoals records, sets, pointers zijn onbereikbaar geworden. Voor het gebruiksgemak wordt dus eigenlijk een te hoge prijs betaald. Hiermee wordt bedoeld dat de elementen die een docent wel in een auteurstaal cq -systeem vindt onderwijskundig gezien op een veel te laag niveau liggen, terwijl het bij specifieke vormen vaak meer moeite kost deze te realiseren dan met een algemene programmeertaal het geval is. In dit laatste geval werkt de auteurstaal de verkeerde kant uit: de bereikbaarheid van een oplossing is voor de docent cq programmeur afgenomen i.p.v. toegenomen. Dit is geen pleidooi tegen auteurstalen! Ik wil er alleen mee zeggen dat een auteurstaal als ontwikkelomgeving alléén een tussenoplossing is, waar een onderwijskundige een verbinding naar 'boven', en een programmeur een verbinding naar 'onderen' in mist, en dat hierin het spanningsveld tussen resp. gebruikersvriendelijk en transparant gelegen is.

3 De workbench

In het bovenstaande hebben we een aantal problemen ten tonele gevoerd m.b.t. de structuur van computerondersteund onderwijs en de wijze van ontwerpen cq bouwen.

Wat we ontberen is een probleemspecifieke ontwikkelmethodiek, specifiek voor het onderwijs en onderwijsleersystemen. Aan een ontwikkelmethodiek kunnen bepaalde eisen gesteld worden. Deze moet minimaal bestaan uit:

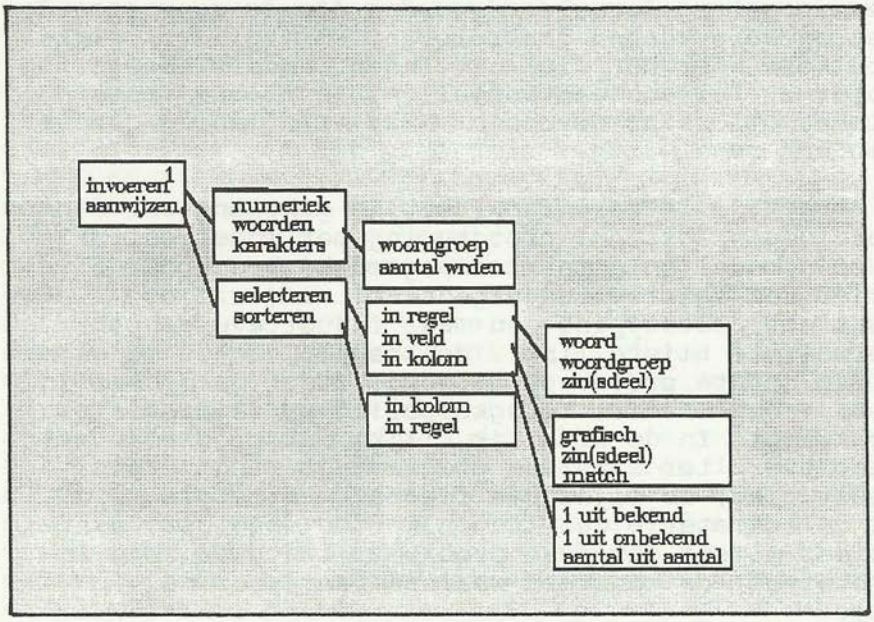
1. technieken (onderling samenhangende procedures) die de docent door het ontwerpproces heen leiden;
2. modellen die, op meerdere detailleringsniveaus, het ontwerp op formele wijze kunnen weergeven;
3. regels met betrekking tot de consistentie van het ontwerp.

Voorts zouden we willen dat een ontwikkelomgeving aansluit bij de (goede) gewoontes van een gebruiker, en dat ook oplossingen geboden worden voor problemen, nog voor deze als zodanig herkend worden; m.a.w. dat de ontwikkeling computerondersteund is.
Zonder adequate ontwerptechnieken en zonder modellen die het ontwerp op formele wijze beschrijven, kunnen ook geen regels geformuleerd worden die de consistentie van het ontwerp kunnen controleren, en mag de ontwikkelaar telkens weer het wiel uitvinden. Anders gezegd: dan wordt er telkens een eenmalige, in plaats van een generieke, oplossing gevonden. Naar mijn mening gebeurt dit dan ook regelmatig.

In de utrechtse letterenfaculteit is gekozen voor een haalbare deeloplossing: prototyping met modellen. Enerzijds zijn modellen gemaakt die goed herbruikbaar zijn (en is er dus van tevoren nagedacht), anderzijds ontstaat met prototyping een meer inductieve aanpak (achterwaarste bijstelling). Met behulp van de modellen wordt een eerste prototype gebouwd, zodat de docent in een zeer vroeg stadium terugkoppeling op zijn/haar ideeën krijgt. In de praktijk blijkt dat problemen met het lesontwerp ter zake van programmasturing versus studentsturing, de wijze van feedback verzorging e.d. vanuit de lesmaterie wel kunnen worden opgelost, echter pas op het moment dat deze problemen tastbaar zijn in het prototype (cq concreet waarneembaar voor de docent).
Bij zo'n modelgestuurde ontwikkelwijze is wel de mate waarin een docent zijn onderwijs in de modellen kan projecteren van belang, plus de eindigheid van de modellenverzameling. De modellen zijn hiertoe op een aantal niveau's gerealiseerd. Het laagste niveau is dat van de afzonderlijke vraagvorm, op het middenniveau bestaan modellen welke de samenhang van afzonderlijke vraagvormen en presentaties tot lessegmenten realiseren, en op het hoogste niveau bestaan modellen die een programmastructuur met gebruikerskoppeling weergeven. Alle modellen zijn verzameld in de 'workbench'.

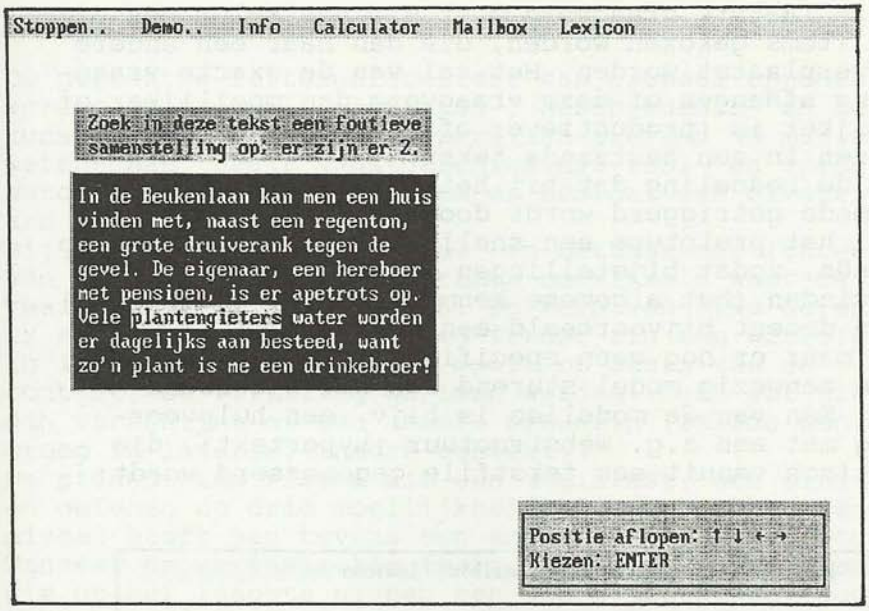
Wanneer een docent bijvoorbeeld een reproductieve leertaak wil realiseren kiest hij/zij een interactiemodel uit de workbench waarvan de vorm als middel past bij het doel en de inhoud. Elk model is een herkenbaar kleinste deeltje onderwijs. Hieruit kan men opmaken dat modelgeoriënteerd ontwikkelen ook pragmatischer, minder precies, is. Ook zal men in een wat complexer lesontwerp nooit 100% dekking met modellen vinden. Naast de voordelen van de werkwijze (herkenbaarheid, snelheid e.d.) zijn er dus ook nadelen.

De modellen op het laagste niveau, de vraagvormen, zijn geordend vanuit het invoertype (hoe en wat). De eerste scheiding hierin is tussen 'invoeren' en 'aanwijzen', resp. overeenkomend met de activiteiten herinneren en herkennen. Door verdere vertakkingen onstaan uiteindelijke in totaal ca. 40 verschillende vraagvormen.
De grove structuur ziet er als volgt uit:



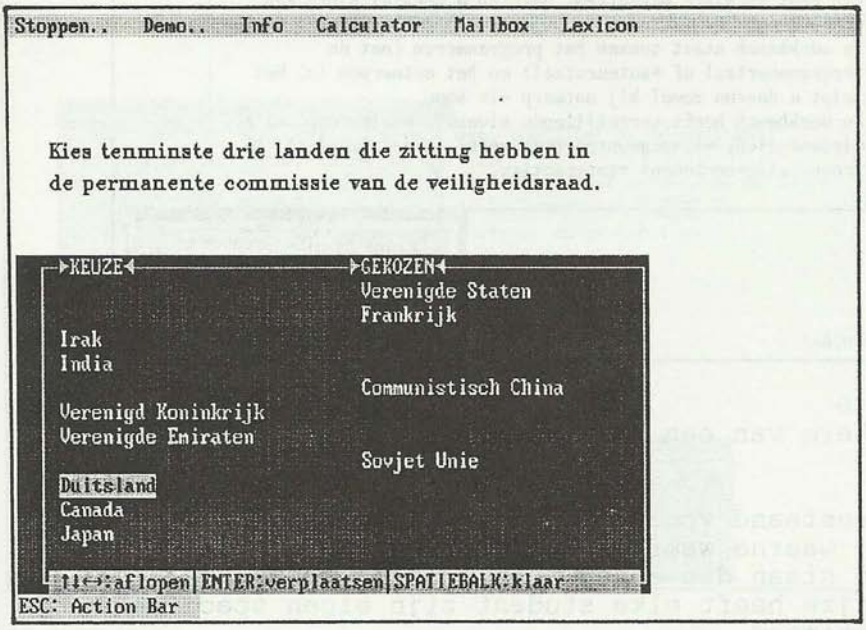
Figuur 2
Globale structuur van de vraagmodellen

Een voorbeeld van een gemengd productieve/reproductieve leertaak is een vraag waarbij een student iets uit een verzameling moet selecteren (herkennen cq aanwijzen). Een scherm van een mogelijke vraagvorm hierbij (selecteren van een zinsdeel uit een tekstblok) kan, uitgaande een voorbeeld nederlands, er zo uitzien:



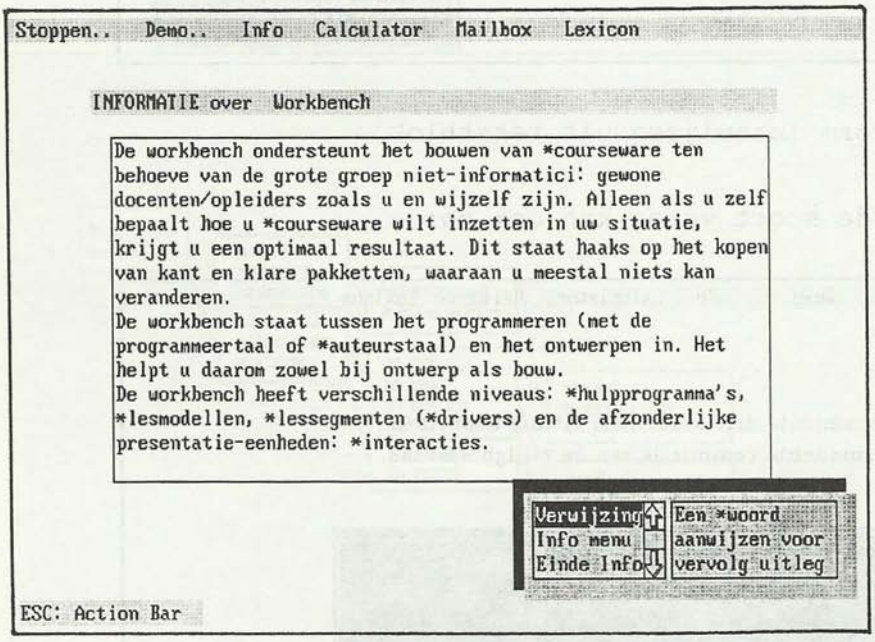
Figuur 3
Vraagvorm 'aanwijzen uit tekstblok'

Dezelfde soort vraag kan ook zo:



Figuur 4
Voorbeeld van een selecteervraag

Bij deze laatste vraagvorm moet uit een kolom een aantal items gekozen worden, die dan naar een andere kolom verplaatst worden. Het zal van de exacte vraagstelling afhangen of deze vraagvorm dan moeilijker of makkelijker is (productiever of reproductiever) dan aanwijzen in een bestaande tekst.
Het is de bedoeling dat bij het ontwikkelen een 'lesidee' mede getriggerd wordt door bestaande modellen, terwijl het prototype een snelle terugkoppeling is op de ideeën, zodat bijstellingen in een vroeg stadium plaatsvinden (het algemene kenmerk van 'prototyping'). Als een docent bijvoorbeeld een hulpdeel wil realiseren, maar er nog geen specifieke ideeën over heeft, kan een aanwezig model sturend dan wel oplossend werken. Eén van de modellen is bijv. een hulpvoorziening met een z.g. webstructuur (hypertext), die automatisch vanuit een tekstfile gegenereerd wordt:

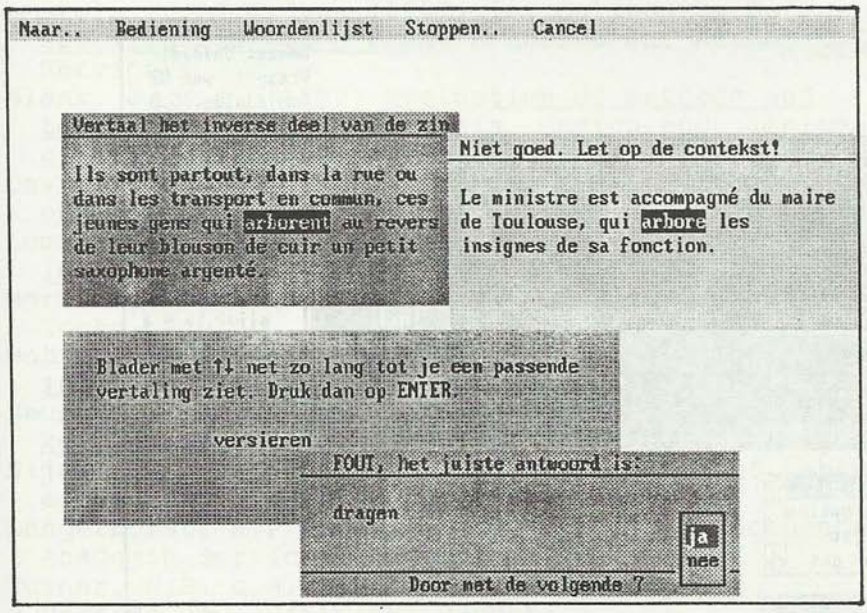


Figuur 5
Hulpscherm van een 'hypertext' hulpdeel

In bovenstaand voorbeeld kan een student een woord aanwijzen, waarna weer een schermpresentatie volgt waarop woorden staan die aangewezen kunnen worden, etc. Op deze wijze heeft elke student zijn eigen specifieke instructiepad.
O.a. omwille van de toegankelijkheid op programmaniveau is de workbench zoveel mogelijk uitgewerkt in TAIGA.

4 Twee voorbeelden uit de praktijk

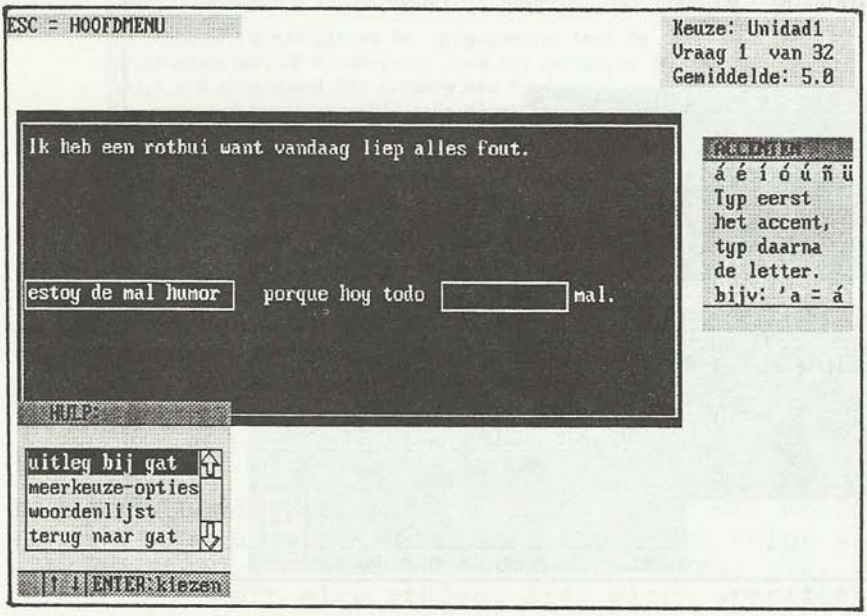
De utrechtse letterenfaculteit kan globaal opgedeeld worden in de taal-, letteren-, geschiedenis- en de kunstvakken. Naast nieuw-perzisch treffen we muziekwetenschap, naast kunstgeschiedenis taal- en spraakautomatisering, etc; kortom er bestaat een diversiteit aan disciplines.
Bij de vakgroep Franse Taal -en Letterkunde (ontwerp T. van Wesemael) is een programma ontwikkeld waar de leesvaardigheid wordt geoefend. De kern van elke oefening is een vraag waarbij in een franse zin een woord(groep) in inverse staat. Hiervan moet, op basis van de context, de vertaling geraden worden. Lukt dat niet, dan verschijnt er een andere steunzin (zelfde woordgroep in inverse, andere context).
De student kan kiezen uit een begintest, een eindtest en oefenen op drie moeilijkheidsniveaus. Elk oefenniveau heeft een tevens een andere feedbackstructuur. Wanneer de vertaalactie faalt komt er een wash back, die op het laagste niveau een soort meerkeuzevraag is. Op het middelste niveau wordt een gatenoefening in de vertaalde tweede contextzin gegeven, en op het hoogste niveau volgt een selecteervraag. Als voorbeeld een scherm van een vraag op het laagste werkniveau, op het moment van de wash back-vraag:



Figuur 6
Woorden vertalen op basis van context

De te raden woorden zelf zijn opgedeeld in probleemcategorieën. Hiermee is een situatie bereikt die overeenkomt met de werkelijkheid: een bepaalde context met woorden die we niet kennen, maar die uit de context afgeleid kunnen worden. Het is natuurlijk mogelijk dat ook de context zelf problemen kan opleveren. Als een woord uit de context onduidelijk is kan de vertaling hiervan direct opgevaagd worden in de online woordenlijst. De structuur van de les is deels programmagestuurd, deels studentgestuurd. De docent heeft op bepaalde momenten op inhoudelijke gronden gekozen voor of het een, of het ander. Tijdens prototyping kwam dit tot stand.

Bij een cursus spaans (vakgroep Spaanse Taal- en Letterkunde, ontwerp R. de Graaff) gaat het om het vertalen van zinsdelen. De les is geheel studentgestuurd, maar programmasturing is consequent toegepast op het niveau van de enkele vraag. Als het vertalen mislukt is er keuze uit een aantal activiteiten, waaronder een meerkeuzevraag. Daarna wordt naar de hoofdvraag teruggekeerd en er is pas voortgang als deze juist beantwoord is. Het probleem van de 'adequate feedback' is opgelost door keuze uit activiteiten (afhankelijk van de informatiebehoefte van de student). Als voorbeeld een scherm van een vraag op het moment van de feedback:



Figuur 7
Fragment uit cursus spaans vertalen

5 Afsluiting

Bij beide cursussen, frans en spaans, is in een vroeg stadium, door de vakgroep Computer & Letteren een prototype vervaardigd. In beide gevallen nam het eigenlijke bouwen hiervan ca. 2 dagen in beslag en kon tot ca. 90-95% dekking gevonden worden uit de bestaande modellen. Het editen van de cursussen kostte uiteraard meer tijd (enkele weken, hierbij wordt gebruikt gemaakt van externe databases); dit werd uitgevoerd door studentassistenten. Het nadenken over wijzigingen zowel door de docent als door Computer & Letteren is ook tijd die meebetrokken moet worden, het is echter moeilijk deze te meten. In alle gevallen werd het prototype meer dan eens bijgesteld, telkens een ½ - 1 dag werk genererend. Al met al bleef de gehele bouwtijd binnen 1 week, kostte de edittijd gemiddeld 3 weken en is enkele dagen besteed aan overleg e.d. De effectief bestede ontwikkeltijd (vanaf 1e gesprek tot en met realisatie) lag bij deze (en andere) projecten rond de 5 weken. Daarbij is courseware gerealiseerd die de student enkele uren aan het werk kan houden.
De workbench wordt op deze wijze ook buiten de letterenfaculteit (o.a. bij technische vakken) gebruikt.

Gebruikte literatuur

Amstel, ir. J.J. van (1987) in: Colloquium Software specificatie technieken. Schoonhoven: Academic Service.

Blank, J. e.a. (1982) Evaluation of methods and techniques for the analysis, design and implementation of information systems. Den Haag: Academic Service.

Davis, W.S. (1985) Inleiding systeemanalyse en systeemontwerp. Den Haag: Academic Service.

Lumsdaine, A.A. (1960) Teaching machines and programmed learning; a source book. Washington : Davi.

Merrill, M.D. (1980) TICCIT. New Yersey: Educational Techn. Publ.

Motill, J. (1984) Programming principles; an introduction. Boston: Allyn and Bacon.

Neuner, G. e.a. (1981) Uebungstypologie zum kommunikativen Deutschunterricht. Berlin: Lamgenscheidt

Nijssen, G.M. (1989) Behoeftebepaling vraagt geheel eigen methodiek. In: Computable 10, 31.

Langerhorst, R.P. (1987) Gegevensanalyse. Schoonhoven: Academic Service.

Turner, W.S. e.a. (1986) System Development Methodology. Rijswijk: Pandata.

Verpoorten, J.H. (1989) Workbench voor het computerondersteund onderwijs. Utrecht: ECCO, RUU.