Gebruikersvriendelijkheid van Geautomatiseerde Systemen I & II

166552 166557

Ben Hekster
Tankelanden 5
7542 DR Enschede
053-764091

Peter Middelhoek
Glanestraat 19
7555 KW Hengelo
074-911674

Die Qualität des Lebens
aus kleinem, intelligentem Element

—*Wirtschaft ist tot,* Laibach [*Kapital*]

Printed on Monday, October 2, 2000, in 9-point New York on the Apple LaserWriter

Created on the Apple Macintosh, using Microsoft Word, Claris MacDraw Pro, Claris MacPaint, Microsoft Excel, Apple ResEdit, Apple TeachText, and the Macintosh Programmer's Workshop

Inleiding

VERANTWOORDING

Wetenschappelijk onderzoek maakt tegenwoordig steeds vaker gebruik van de computer, bijvoorbeeld door het gebruik van simulaties. En dit soort onderzoek wordt vaak verricht met behulp van zeer taak-specifieke software, waarvan de ontwikkeling aanvankelijk veelal is begonnen in een tijd waarin gebruikersvriendelijkheid een nogal onderschikte rol speelde aan bijvoorbeeld, programmeursvriendelijkheid of resourceverbruik en performance van de software. Met name heeft dit tot gevolg dat een gebruiker van deze software erg veel tijd moet investeren in het leren en gebruiken van wat uiteindelijk slechts hulpmiddelen zijn bij het verrichten van zijn eigenlijke taak.

Inderdaad heeft de vergaande ontwikkeling van geavanceerde hardware en software het nu mogelijk (en vanwege de toenemende complexiteit vaak zelfs nodig) gemaakt dat veel meer aandacht wordt besteed aan de *bruikbaarheid* van softwareapplicaties. Specifieke toepassingen kunnen dan toegankelijker worden voor een veel groter publiek dan anders het geval was.

Slechts een luxe is dit niet, wat onder meer blijkt uit persoonlijke ervaringen van beide auteurs en die van anderen. Het verslag van [Remmers90] maakt bijvoorbeeld duidelijk dat zelfs voor kundige studenten het een niet voor de hand liggende taak is om de voor hen essentiële hulpmiddelen te gebruiken (in dit geval ging het om het schrijven van simulatiespecificaties voor de processimulator SUPREM3). In ogenschouw nemend de moeilijkheden die zelfs ondervonden worden door ervaren gebruikers, kunnen we concluderen dat deze programma's niet-triviale problemen veroorzaken. Tekstspecificaties zoals die in de UNIX-cultuur zo populair zijn [Bell, section 5], zijn vaak bijzonder lang en ondoorzichtelijk. Deze wordt dan meestal alleen door de schrijver zelf nog enigszins begrepen, en dan nog slechts voor korte tijd. Dit zijn voorbeelden van problemen in de *hoge-niveau* of structurele specificatie.

Verder maakt de ondoorgrondbaarheid van de specificatie het nog steeds te gemakkelijk om typfouten of andere *lage-niveau* specificatiefouten te maken. In dit laatste geval heeft de gebruiker zelf nog wel begrepen wat hij doet maar is hij niet in staat deze kennis op een voldoende nauwkeurige manier voor het programma te formaliseren.

Omdat al deze fouten vaak pas veel later na een aanzienlijke hoeveelheid processing door het programma worden gevonden zien we dat het gebruik van zo'n hulpmiddel al snel elke vorm van interactiviteit verliest en in plaats daarvan de gestalte begint aan te nemen van *batch-processing*.

PROBLEEMIDENTIFICATIE

Het probleem van de bruikbaarheid van complexe software werd aan het de eind van de jaren 80 in de vakgroep ICE van de faculteit Electrotechniek van de Universiteit Twente geïdentificeerd, waar de IC-fabricageproces- en componentsimulatiesoftware (process and device simulation software) "TRENDY" [Schie90, Wolbert91] ontworpen en in gebruik is. Er waren in principe twee manieren waarop het probleem zou kunnen worden zijn aangepakt:

- opnieuw ontwerpen, schrijven, testen en debuggen van de simulator;
- ontwikkelen van nieuwe software die samenwerkt met de bestaande simulator.

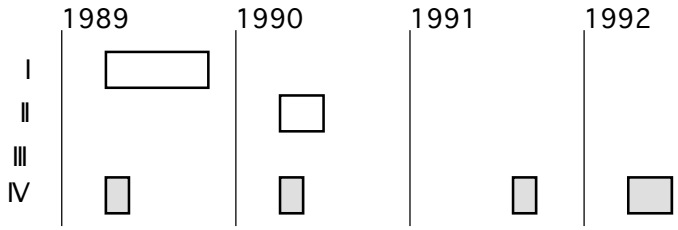
De eerste optie leidt tot een beter en meer geïntegreerd produkt, maar heeft een aantal grote nadelen. Ten eerste zijn gespecialiseerde softwarehulpmiddelen vrijwel altijd het resultaat van vele tientallen manjaren onderzoek in algoritmes en ontwikkelingswerk, door zeer gespecialiseerde programmeurs— het is duidelijk dat het zeer onwenselijk zou zijn om dit werk te moeten dupliceren. Ten tweede zijn de specifieke programma's door de jaren over de wereld verspreid en zodoende vaak een soort informele standaard geworden. Ontwikkelen van een nieuw programma zou de nieuwe specificaties incompatibel en onuitwisselbaar maken met die van anderen. Tenslotte wilde men een oplossing op relatief korte termijn.

Begin 1989 werd zodoende besloten tot het initiëren van een serie 250-uurs opdrachten om het genereren van invoerspecificaties voor de bestaande TRENDY software te vereenvoudigen. Hierbij is gekozen voor een oplossing die meer algemeen toepasbaar is dan alleen voor TRENDY. In [Middelhoek89a] is een meer gedetailleerde verantwoordig voor het ontwerp gegeven.

Het is interessant zich hierbij te realiseren dat zelfs in die tijd de toepassing van grafische user-interfaces door een groot deel van de computer-gebruikende ingenieurs wereld als kinderachtig en onnodig beschouwd werd. Dit is vermoedelijk mede te wijten aan effecten die optreden wanneer eenmaal de aanzienlijke intellectuele investering is gemaakt om het programma te leren gebruiken: ten eerste, een bepaalde weerstand om de bestaande vergaarde kennis op te geven (een soort "cognitieve inertie"); maar ook een bepaalde minachting voor mensen die niet bereid zijn dezelfde investering te maken. De huidige situatie is heel anders en grafische gebruikersinterfaces worden steeds meer als essentiëel onderdeel van computersystemen gezien, vooral door mensen die met grafische interfaces zijn opgegroeid. Dit heeft onder meer tot gevolg gehad dat gedurende de afgelopen jaren veel energie is gestoken in de standaardisatie van grafische interface *toolkits*. Helaas was dit in de tijd dat dit project startte nog niet het geval. In het bijzonder was OSF/Motif nog 'vaporware' en het grafische user-interface zoals gebruikt in de Apple Macintosh eigenlijk de enige stabiele en complete toolkit.

ONTWIKKELING VAN TRIP

Het in dit verslag beschreven werk vormt de vierde fase van de ontwikkeling van *TRIP*, de TRENDY Input Processor. TRIP is een programma dat het maken van invoerspecificaties voor TRENDY drastisch moet gaan vereenvoudigen en versnellen.



Figuur 1. Ontwikkeling van TRIP

In maart 1989 begon Peter Middelhoek de ontwikkeling van TRIP. Er werden globaal vier fasen in de ontwikkeling geïdentificeerd:

- *Fase I* Ontwerp van het TRIP systeem en een deel van de implementatie (Command en Library Managers— Peter Middelhoek [Middelhoek89a])
- *Fase II* Ontwerp en implementatie van het dialooggeneratie-systeem (Dialog, Editor en Layout Managers— Ben Hekster [Hekster90])
- *Fase III* Ontwerp en implementatie van het commando-systeem (Structure Manager)
- *Fase IV* Ontwerp en evaluatie van het user-interface (Ben Hekster en Peter Middelhoek)

Het tijdsverloop van de ontwikkeling van het project is in het volgende diagram weergegeven:

De vierde fase, beschreven in dit verslag, zou beëindigd worden met de evaluatie van het user-interface na de voltooiing van de implementatie. Aan het begin van het project werd met dhr. Bakker hiervoor een opdracht gespecificeerd [Bakker89] die uitgevoerd wordt in het kader van de W&M themavakken *Gebruikersvriendelijkheid van Geautomatiseerde Systemen I* en *II.*

De derde fase van de implementatie zou uitgevoerd worden na voltooiing van de eerste twee, tussen juni 1990 en februari 1991. Helaas blijkt het voor de vakgroep ICE sinds deze periode niet mogelijk geweest te zijn een student bereid te vinden om deze taak op zich te nemen. Hierdoor, en door een zeer ongelukkig samenspel van onderlinge afwezigheid van beide auteurs ten gevolge van respectievelijke bedrijfsstages in de Verenigde Staten, en vervolgens de afstudeeropdrachten waarvan er één in België, is de vierde fase steeds verder vooruit geschoven. Tot op vandaag de dag is er *nog steeds* geen 250-uurs opdracht uitgevoerd om het derde-fase werk te verrichten.

Verder is het zo dat de oorspronkelijke leiders van het project, Philip Wolbert en Eddie van Schie beide niet meer in dienst zijn van de UT.

Mede als gevolg hiervan heeft men de softwareomgeving waarin TRIP ontwikkeld is zover laten veranderen dat essentiële delen van de TRIP gebruikersinterface modules niet meer gecompileerd kunnen worden. In het bijzonder gaat het hier om de overschakeling van de Xr Toolkit naar de OSF/Motif Toolkit [OSF90a, OSF90b]. Dit zijn libraries die extra functionaliteit leveren boven het X Windows systeem zelf, die betrekking hebben tot zeer essentiële dialoogfunties. Alhoewel de dialoogfuncties van TRIP genoeg modulair zijn gebouwd dat een herimplementatie in theorie tot de mogelijkheden zou behoren, hebben wij het niet haalbaar geacht om dit zelf nog in enig redelijk tijdsbestek te verrichten, met name omdat wij geen van beide voldoende vertrouwd zijn met de nieuwere OSF/Motif Toolkit.

Wij vonden onszelf aldus in de uiterst benarde positie dat we het user-interface moesten evalueren van een incomplete implementatie. Omdat deze vierde fase absoluut niet nog verder uitgesteld kon worden was het noodzakelijk dat we zelf nog een zeer aanzienlijke hoeveelheid extra software moesten schrijven om alsnog deze evaluatie uit te kunnen voeren. Wij hebben daarbij gekozen voor het implementeren van een

prototypeprogramma dat veel van het gebruikersinterface en de inwendige functionaliteit van TRIP simuleert. De implementatie hiervan vindt plaats op Apple Macintosh computers, onder meer omdat deze voorzien zijn van zeer substantiële ingebouwde user-interface ondersteuning in de systeemsoftware, en omdat dit de enige mogelijkheid was om in enigszins redelijke hoeveelheid tijd een functionerend programma te kunnen realiseren. Dit prototype bestaat uit ongeveer 9,000 regels C++ code wat overigens overeen komt met het aantal regels programmacode in de oorspronkelijke TRIP software.

Merk overigens op dat het in de softwareindustrie geen uitzondering is om, vooral bij grotere projecten, zelfs nog tijdens de ontwerpfase van de software een 'mock-up' van het user-interface te maken en te laten evalueren. Zo'n evaluatie kan van groot belang zijn bij het verifiëren van bepaalde aannames betreffende de perceptie van gebruikers van het programma, voordat grote hoeveelheden tijd en geld geïnvesteerd zijn in de werkelijke implementatie. Deze ontwikkeling geeft op zich al aan dat gebruikers-interfaces in de industrie een veel meer prominente betekenis beginnen in te nemen.

PROBLEEMIDENTIFICATIE

De volgende problemen in het gebruik van TRENDY zijn geïdentificeerd:

- de specificatiesyntax (losstaand van de semantiek) is te steil
- de specificatie is computer- en niet gebruiker-georiënteerd, zodat de inherente logische structuur slechts moeilijk hierin is terug te vinden
- fouten in de semantiek van de specificatie zijn moeilijk te corrigeren
- fouten in de specificatie zelf worden pas bij het simuleren gevonden

OPBOUW VAN HET VERSLAG

De fase IV opdracht is verdeeld in twee sub-fasen:

- analyseren en specificeren van eisen en richtlijnen waaraan het user-interface behoort te voldoen om bovenstaande problemen te addresseren (IVa)
- evalueren in hoeverre de implementatie hieraan voldoet (IVb)

IVa is grotendeels aan het begin van het project voltooid voordat aan de implementatie was begonnen, en wordt in het eerst volgende hoofdstuk beschreven. Tijdens en na de fase één en twee implementaties, waarvan relevante aspecten in het hoofdstuk daarna worden beschreven, zijn deze nog aangescherpt, gedetailleerd en uitgebreid. Het daarop volgende hoofdstuk beschrijft de TRIP 'mock-up' en geeft de recente evaluatie van het TRIP systeem in fase IVb. In het laatste hoofdstuk van dit verslag worden tenslotte enkele conclusies getrokken en suggesties gedaan voor toekomstig werk en wordt een terugblik op dit werk gegeven.

LITERATUUR

Buiten de in dit verslag gegeven referenties bestaan er nog een aantal die de auteurs als belangrijk beschouwen en die als basis voor het ontwerp van TRIP hebben gediend. Met name denken we hierbij aan het pionierswerk dat in de vroege 80-er jaren door het Xerox PARC instituut is verricht en onder meer heeft geresulteerd in Smalltalk [Goldberg84] en het bijbehorende programmeursinterface. Verder wordt verwezen naar [Middelhoek89a] en [Middelhoek89b] waarin zich meer uitgebreide bibliografieën bevinden, die naast algemene artikelen over gebruikersinterfaces met name veel artikelen bevatten die het X Window systeem behandelen. Ook verschijnen er de laatste tijd veel artikelen die nieuwe methoden van omgaan met computers beschrijven, zoals bijvoorbeeld de hoogst interessante [Weiser91] en [Marcus91]. Deze verschaffen veel inzicht in het ontwerpen van moderne gebruikersinterfaces.

Principes van User-Interfaces

Dit hoofdstuk beschrijft op welke wijze de problemen die in het vorige hoofdstuk geïdentificeerd zijn opgelost kunnen worden. Bij deze analyse hebben we ons vooral laten leiden door [Apple86], [Apple85 vol. I ch. 2, vol. IV ch. 1, vol. V ch. 2, vol. VI ch. 2], [Foley90 ch. 9.3], en in mindere mate door [Buurman85]. Met name de literatuur van Apple en Foley worden wereldwijd als toonaangevend beschouwd. [Buurman85] is voor ons doel minder geschikt aangezien de hoofdstukken die gewijd zijn aan de gebruikersvriendelijkheid van programmatuur een generatie software beschrijven die nu duidelijk verouderd is. Gelukkig zijn veel van de beschreven algemene begrippen wel toepasbaar. Deze literatuur geeft behalve algemene richtlijnen ook zeer gedetailleerde methoden voor het ontwerpen van een user-interface en de vele elementen daarin.

Hierbij identificeren we een aantal basisprincipes die we voor de implementatie van belang achtten. Omdat de *dialogische*, *menu*, en *iconische representatie* van informatie in het user-interface zo'n belangrijke plaats innemen werken we de hier boven gerefereerde basisprincipes voor deze drie nog in meer detail uit.

PRINCIPES

Uit de literatuur en met de door ons zelf opgedane ervaring willen we hier een aantal basiselementen van user-interface ontwerp naar voren halen. In de volgende sectie zullen deze principes één voor één besproken worden. In de latere hoofdstukken zullen we van deze principepes gebruik maken om het ontwerp van TRIP te analyseren.

Gebruik metaforen. Gebruikers zijn in het algemeen geen ervaren programmeurs. Ook zijn de meeste programmeurs geen ervaren gebruikers van het programma. Metaforen maken gebruik van de al bij de gebruikers aanwezige ervaring om de werking van het programma op een intuïtieve manier duidelijk te maken. Het stelt ze

dus in staat om bepaald gedrag te voorspellen. Bij het kiezen van een geschikte metafoor moet men zich wel afvragen of het er een is die bij gebruikers van het programma bekend is.

Directe manipulatie. Als het niet mogelijk is instantaan een gewenst resultaat te bewerkstelligen is metafoor-manipulatie een effectief surrogaat. Het is veelal zo dat simulatie te lang duurt om het gevolg van een verandering in de specificatie direct weer te kunnen geven. In plaats daarvan verandert de schematische representatie.

'Kijk en wijs' (niet: 'onthoud en typ'). Dit ligt ten grondslag aan het TRIP systeem: in plaats van het moeten leren en onthouden van een hele commandosyntax met inbegrip van toegestane opties en sub-opties voor elke simulator die gebruikt wordt, voert de gebruiker een op een consistente wijze gepresenteerde dialoog waarin commando's en opties al zichtbaar zijn. Het blijft dan slechts nog een zaak van opties selecteren of gevraagde informatie leveren.

Consistentie. Het is uiterst verwarrend wanneer twee programma's op hetzelfde systeem, of zelfs verschillende delen binnen één programma, een zelfde actie radicaal verschillend interpreteren [Anonymous90]. Consistentie geeft in nieuwe en onvertrouwde situaties een vertrouwde en suggestieve indruk.

Door gebruik van gestandaardiseerde grafische systemen wordt gepoogd de inter-applicatieve consistentie te waarborgen. Het MIT X Window systeem dat zich nu als een *de facto* standaard heeft ontwikkeld garandeert compatibiliteit van de software tussen verschillende systemen en een minimale vorm van gebruiksconsistentie. Het gestandaardiseerde grafische interface staat het gebruik toe van andere grafische user-interface*toolkits* die de consistentie van het user-interface tussen verschillende programma's waarborgen.

Vooral bij de metafoorkeuze is het belangrijk dat deze consistent is, omdat deze fundamenteel is voor het gehele interface. Twee verschillende metaforen binnen één programma (of zelfs binnen één systeem), die afzonderlijk goed werken, kunnen zeker voor beginnende gebruikers verwarrend zijn wanneer ze door elkaar gebruikt worden [Apple89 card 3]. In de praktijk betekent het dat de gebruiker de keuze heeft om alle verschillen uit het hoofd te leren (wat in het gebruik vertragend werkt) of om de functie nergens te gebruiken.

Twee interessante specifieke voorbeelden van inconsistenties binnenin Microsoft Windows worden gegeven in [Anonymous90]:

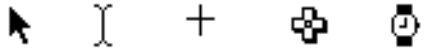
Het eerste betreft het Windows desktopmetafoor. Als een al eerder geopend programma nog eens wordt geopend, betekent dit soms "breng het al geopende programma naar voren" (bijv. WingZ) en soms "open hetzelfde programma opnieuw" (Excel). Dit zijn twee totaal verschillende acties.

Consistentie heeft zeker ook betrekking op de functie van specifieke toetsen. Een goed voorbeeld hier is in het geval van drie Microsoft programma's voor Windows. Wanneer een woord geselecteerd is en op "delete" gedrukt wordt, en dit in PowerPoint betekent "delete het hele woord", in Excel "delete het laatste karakter", en in Word "delete niets, maar ga naar het begin van het woord", dan verstoort dit volledig het intuïtieve gebruik.

"What you see is what you get". Dit principe stelt dat de schermrepresentatie van informatie en afgedrukte representatie overeen dienen te komen.

In TRIP is dit nog niet van belang aangezien het onder normale omstandigheden geen hard-copy genereert. Men zou dit begrip echter ook ruimer kunnen opvatten door nadruk te leggen op de consistentie tussen de invoer en de door de simulator uitgevoerde simulatie.

User control. Het is van belang dat de gebruiker het gevoel heeft dat *hij* de computer bestuurt en niet dat hij door de computer bestuurd wordt.



Figuur 1. Voorbeelden van cursorvormen

Figuur 2. Grayscale icoon en gesimuleerde grayscale-effecten in zwart/wit

In het geval van TRIP betekent dit bijvoorbeeld dat de gebruiker het initiatief neemt tot interactie.

Feedback. Gebruikers willen zien dat hun acties een gevolg hebben, geef ze dus altijd voldoende feedback. Een goede toepassing van dit principe is in het maken van *selecties*. In grafische metafoorsystemen is de manier waarop de gebruiker een bepaald resultaat bewerkstelligt meestal een twee-staps proces: eerst wordt aangegeven *waarop* de actie betrekking moet hebben door middel van een selectie, en pas ten tweede wordt aangegeven *wat* de actie is. Het is daarom belangrijk dat het ten allen tijde direct duidelijk is wat de selectie is.

Over de manier waarop dit het meest effectief gedaan kan worden is nog wel wat te zeggen. Vaak wordt dit gedaan door middel van het grafisch inverteren van de visuele representatie van de selectie. Voor zwart/wit-systemen is de betekenis van het inverteren duidelijk— zwarte pixels worden wit, en omgekeerd. Op kleursystemen is dit echter niet direct duidelijk. Een stricte wiskundige generalisatie van “inverteren” naar kleuren is het complementeren van de RGB-componenten. Dit geeft echter meestal bizarre resultaten, en in bepaalde gevallen is een dusdanig geïnverteerde kleur niet of nauwelijks van zichzelf te onderscheiden. Een betere methode om de geselecteerde toestand aan te geven om in het icoon de achtergrondkleur te veranderen in een specifieke *selectiekleur*. Uiteraard is het wenselijk dat de gebruiker zelf deze selectiekleur kan kiezen, en liefst dat deze ten bate van de consistentie in alle programma's hetzelfde is.

Een andere, meer subtiele maar zeer effectieve, methode van feedback is het veranderen van de vorm van de cursor. Deze geeft in één oogopslag informatie over de functie van het schermgebied waarboven de cursor zich bevindt. Omdat gebruikers over het algemeen snel met de muis de verschillende schermgebieden kunnen bereiken zijn ze hierdoor snel in staat een ‘eerste-orde’ indruk van de windows in het programma te verkrijgen. In de Macintosh zijn een aantal verschillende cursorvormen gedefiniëerd, zoals de standaard “arrow pointer”, “I-beam” voor tekstselectie, “cross beam” voor grafische selectie, “plus” voor selecties van blokvelden (zoals in een array), en de “watch” cursor om aan te geven dat een langurende operatie gaande is:

Vergevendheid. Voorkom dat kleine fouten van de gebruiker grote negatieve gevolgen hebben, immers gebruikers kunnen zich vergissen. Zorg ervoor dat de gebruiker in zulke gevallen de mogelijkheid heeft fouten te corrigeren.

Een actie van de gebruiker die essentiële informatie verandert moet ongedaan gemaakt kunnen worden door middel van ‘undo’ of ‘cancel’ functies. In het bijzonder, als veranderingen die gemaakt worden aan de informatie in een dialoog bij nader inzien toch niet juist zijn, kunnen ze worden ‘vergeten’ door met de muis in de “Cancel” button te klikken.

Als een undo-functie niet mogelijk is, moet de gebruiker van tevoren gewaarschuwd worden dat dit het geval is.

Stabiliteit. Gebruikers voelen zich meer gerust in een omgeving die herkenbaar blijft dan een die bij elke actie visueel radicaal verandert.

Esthetische integriteit. De gedachte, dat een gebruiker die in bezit is van een computersysteem dat 2^{10} kleuren ondersteunt deze ook allemaal tegelijk wil zien is een misconceptie van sommige programmeurs [Microsoft89]. Kleur moet gebruikt worden als *ondersteuning*, om betekenis aan het interface toe te voegen, en zelden omdat ze er gewoonweg leuk uitzien. Men loopt anders het risico dat het interface er overweldigend of 'kinderachtig' uit gaat zien ("angry fruit salad" [Hacker91]). Het mag nooit zo zijn dat kleur de enige manier is om verschillende elementen van het interface van elkaar te onderscheiden. Dit verzekert dat het interface voor gebruikers van monochrome systemen, onder slechte verlichtingsomstandigheden of met bepaalde vormen van kleurenblindheid, nog steeds volledig toegankelijk is. Uit onderzoek blijkt namelijk dat 8% van mannen en 0.5% van vrouwen één of andere vorm van kleurenblindheid heeft.

Één methode om hiervan verzekerd te zijn is om het interface *eerst* in zwart/wit te ontwerpen, en dan later pas kleur toe te voegen. Helaas is er bij programmeurs een sterke neiging om dit precies andersom te doen. Uiteraard moeten de kleur- en de zwart/wit-versies van het interface niet in al te sterke mate verschillen.

Verder moet ook voorzichtig omgegaan worden met het 'simuleren' van kleureffecten in zwart/wit. Het is bijvoorbeeld onverstandig om kleur- of grayscale-'schaduweffecten' met behulp van patronen in zwart/wit trachten na te doen. Dit voegt onnodige 'informatieruis' aan de representatie toe en vermindert daarmee de herkenbaarheid.

Voorkom 'mode's. Een 'mode' is een plaats in het programma dat expliciet in- en uitgetreden moet worden en waar de context waarbinnen de gebruiker werkt verandert. Concreet zou dit bijvoorbeeld tot uiting kunnen komen doordat het scala van mogelijke gebruikersacties is beperkt. Omdat mensen in het algemeen in werkelijkheid ook niet 'modaal', dat wil zeggen in moden, werken noch zich continu bewust willen zijn van de context waarin het programma zich bevindt, komt deze vorm van interactie uiterst onnatuurlijk en onvriendelijk over.

Voorkom ingebouwde afhankelijkheden. De software zelf moet zo min mogelijk afhankelijk zijn van de gebruikte hardware en software, zodat het zonder problemen op verschillende computers gebruikt kan worden. Zo moeten in het programma geen verborgen afhankelijkheden zijn ingebouwd van grootte of aantal schermen, of het aantal of soort kleuren, type toetsenbord, etc.

Hergebruik. In de kringen van software engineering is het begrip *reusability* allang doorgedrongen, maar heeft zeker zijn toepassing in user-interfaces. Het hergebruiken van informatie maakt het onnodig de informatie opnieuw te genereren wat tot een aanzienlijke tijdsbesparing kan leiden. Tevens hoeven uit de invoer niet opnieuw fouten gehaald te worden aangezien dit in de eerdere generatie fase reeds gebeurd is. Ook dit kan de gebruiker een aanzienlijke hoeveelheid tijd besparen, zeker als men beseft dat tijdens software ontwikkeling wel 50 procent van de tijd besteed wordt aan het vinden van fouten. Tenslotte is herhaald invoeren van dezelfde informatie voor de gebruiker in het algemeen zeer frustrerend.

Minimaliseer geheugengebruik. Een goed gebruikersinterface moet het geheugen van de gebruiker zo min mogelijk belasten. Dit kan onder meer gebeuren door gebruik te maken van *herkenning* in plaats van *herrinnering*. Op verschillende plaatsen in TRIP wordt hiervan gebruik gemaakt.

Het begrip **symmetrie**, zoals gehanteerd in [Buurman85], kan ondergebracht worden bij het principe van *consistentie*. Overigens zij opgemerkt dat door het niet gebruiken van modes veel van de situaties waarin symmetrie belangrijk is überhaupt niet meer voorkomen. Hierbij kan bijvoorbeeld gedacht worden aan plaatsen waar sprongen in de dialoogstructuur gemaakt worden.

DIALOGISCHE REPRESENTATIE

Deze vorm van communicatie vormt een essentiëel onderdeel van elk grafisch gebruikersinterface. Alle van de eerder opgesomde principes komen terug in deze vorm van interactie. Met name begrippen als 'mode' en 'vergevendheid' verdienen hier extra aandacht.

Een dialogische representatie wordt gebruikt om de gebruiker de mogelijkheid te geven bepaalde opties van het programma te specificeren die invloed hebben op de werking van het programmma. Het is verleidelijk om voor deze vorm van interactie een systeem met verschillende modes te gebruiken. De ene mode laat het toe veranderingen aan het document aan te brengen waarbij de tweede mode wordt binnengegaan als de gebruiker additionele informatie wil specificeren. In TRIP is deze gemode opbouw voorkomen door het mogelijk te maken dat tegelijk aan het hoofddocument als ook aan de additionele informatie gewerkt kan worden. Er wordt hierbij consistent gebruik gemaakt van de buttons 'OK' en 'Confirm' om de ingevoerde gegevens aan het hoofddocument bekend te maken.

Aan de eis van vergevendheid wordt voldaan door middel van 'Cancel' en 'Revert' mogelijkheden die het mogelijk maken (foute) acties in de dialoog door de gebruiker ongedaan te maken.

MENUREPRESENTATIE

In grafische interfaces wordt veelvuldig gebruik gemaakt van menustructuren. Deze menus zijn veelal van het pop-up type en hebben een *mode* loze structuur— dat wil zeggen dat ze slechts 'on demand' verschijnen en daardoor alleen schermoppervlak en de aandacht van de gebruiker in beslag nemen wanneer hij ze nodig heeft. Aan de andere kant is het wel zo dat menu's *context-gevoelig* zijn omdat op elk punt in tijd alleen die menu's en menu items beschikbaar zijn die dan ook van toepassing zijn. Dit is een groot voordeel boven het altijd presenteren lange van lijsten van alternatieven waar doorheen gescrold en gezocht moet worden.

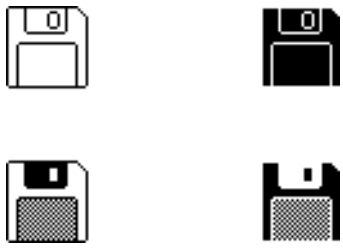
Menus worden gebruikt om de commando's die het programma accepteert op een snelle en eenvoudige manier beschikbaar te maken, zonder dat de gebruiker eerst een aanzienlijke hoeveelheid tijd hoeft te investeren in het uit het hoofd leren ervan. Een groot voordeel is dat veel *type* fouten voorkomen kunnen worden door menus te gebruiken. Maar het belangrijkste aspect, zeker voor beginnende gebruikers, is de beperking van de tijd die initiëel geïnvesteerd behoeft te worden om het programma te kunnen gebruiken, en voor vrijwel alle gebruikers de verminderde belasting van het geheugen.

Voor commando's in menus geldt in het algemeen dat de meest gebruikte bovenaan in de menu behoren te staan, waar ze het snelst geselecteerd kunnen worden. Evenzo worden de 'gevaarlijke' commando's onderin geplaatst om de kans dat de gebruiker per abuis zo'n selectie maakt te minimaliseren.

Hierarchische menu's (ook wel *submenu's*) dienen alleen gebruikt te worden voor *sterk* gerelateerde functies. Dit voorkomt dat gebruikers telkens alle submenu's door moeten bij het zoeken naar een bepaald commando. Het moet uit de 'titel' van het submenu duidelijk zijn welke functies erdoor worden ondergebracht. Aangezien het maken van een submenu-selectie over het algemeen moeilijker is en meer tijd kost, moeten ze spaarzaam gebruikt worden. [Apple85 V-24] adviseert om niet meer dan één niveau te gebruiken. Hierbij dient opgemerkt te worden dat dit betekent dat er feitelijk drie hierarchische niveaus van menuselectie zijn (item in de menubalk, item uit een menu en een selectie uit het submenu)

Veel van de voordelen die geassociëerd worden met menus zijn terug te voeren tot het gebruik van *recognition memory*, waarbij de gebruiker slechts visuele informatie in de vorm van tekst of iconen met bekende woorden en betekenissen hoeft te *associëren*. Dit is in tegenstelling tot het gebruik van *recall memory* in commando-georiëenteerde talen

Figuur 3. Iconen die de 'lege' en 'niet-lege' toestand van de *trash* representeren



Figuur 4. Juist en onjuist ontwerp van een icoon

waarbij de gebruiker zelf een commando of concept uit zijn of haar geheugen moet ophalen.

ICONISCHE REPRESENTATIE

Belangrijke aspecten die bij het ontwerpen van een iconische representatie in het oog moeten worden gehouden [Foley90 p.398]:

Herkenning. Hoe snel en goed de betekenis van een icoon herkend kan worden.

Herinnering. Hoe goed de betekenis van een icoon herinnerd wordt nadat het eenmaal geleerd is.

Discriminatie. Hoe goed verschillende iconen van elkaar onderscheiden kunnen worden.

We kunnen twee hoofdcategorieën van iconische representatie identificeren. In de eerste plaats kunnen iconen gebruikt worden om objecten te representeren. Ze kunnen dan steeds aangepast worden om de huidige status van het object te representeren. Een goed voorbeeld hiervan is de *trash* in het Macintosh desktop metafoor, waarvan de status 'leeg' of 'niet-leeg' visueel wordt weergeven door het gebruikte icoon. De tweede categorie van iconen is de representatie van *acties* of *commando's*.

Er zijn verschillende manieren om te komen tot een keuze van representatie van iconen. In de eerste plaats kunnen de commando's of acties worden gerepresenteerd door iconen die *objecten* uit het dagelijkse leven voorstellen die gebruikt worden voor het uitvoeren van de acties. Een voorbeeld hiervan is het gebruik van een octogonaal "Stop" verkeersbord om het beëindigen van een operatie voor te stellen. Dit sluit aan bij het gebruik van metaforen in het interface. Een mogelijk nadeel hiervan is dat deze iconen dan een twee-staps herkenningproces vereisen: eerst moet het object dat de icoon *voorstelt* worden herkend en dan moet de *functie* van het object worden afgeleid. Dit is een mogelijke oorzaak van problemen.

In het geval van iconische representatie van commando's is een tweede methode die gebruikt kan worden de icoon de situatie *voor en na* uitvoering van het commando te laten representeren. Een potentiëel probleem is dat de icoon een specifiek voorbeeld van

het commando representeert, waardoor het voor de gebruiker onduidelijk kan zijn dat een commando meer algemeen toepasbaar is.

De laatste methode maakt gebruik van *abstracte representaties*. Hierbij kan bijvoorbeeld gedacht worden aan het gebruik van "×" voor delete of een "∨" voor insert. Alhoewel deze vaak bij programmeurs erg populair zijn omdat deze vertrouwd zijn met dit soort abstracties, is dat bij gebruikers in het algemeen niet het geval.

Verder zijn er nog een aantal dingen die bij het ontwerp in het oog gehouden moeten worden. Het geselecteerd zijn van een icoon op een zwart/wit-systeem kan op een goede manier aangegeven worden door het 'inverteren' van de pixels van het icoon. Om dan een duidelijk onderscheid te behouden met de niet-geselecteerde toestand is het noodzakelijk dat in het niet-geselecteerde icoon minder dan 50% van de pixels zwart zijn (zie figuur).

In het algemeen geldt voor iconen dat hoe minder onnodige grafische elementen ze bevatten, des te beter.

OPMERKINGEN

Naast de hier besproken onderwerpen zijn er nog een aantal andere user-interface aspecten die we zeker het onderzoeken en implementeren waard vinden maar waar we helaas wegens tijdgebrek geen of slechts zeer beperkt aandacht hebben kunnen besteden.

We hadden bijvoorbeeld ook graag het gebruik van auditieve hints willen onderzoeken. In het algemeen gaat het gebruik van geluid nooit verder dan een onvriendelijke waarschuwing wanneer een invoerfout wordt gemaakt. Het lijkt ons dat geluid bijvoorbeeld ook gebruikt zou kunnen worden om de overgang tussen noodzakelijke programmamodes aan te geven. Helaas is het gebruik van geluid in de huidige generatie workstations waar wij mee te maken hebben zeer slecht ontwikkeld, zodat we hier geen mogelijkheden tot onze beschikking hadden. Gelukkig zal in deze situatie waarschijnlijk spoedig verandering komen, nu bedrijven zoals Silicon Graphics en Next workstations op de markt brengen die standaard over zeer geavanceerde geluidsmogelijkheden beschikken.

Ook het gebruik van kleur verdient zeker meer aandacht dan we er helaas nu aan hebben kunnen geven. De huidige hardware biedt in veel gevallen ook uitgebreide mogelijkheden hiertoe. Verder onderzoek is zeker gewenst.

Applicatie

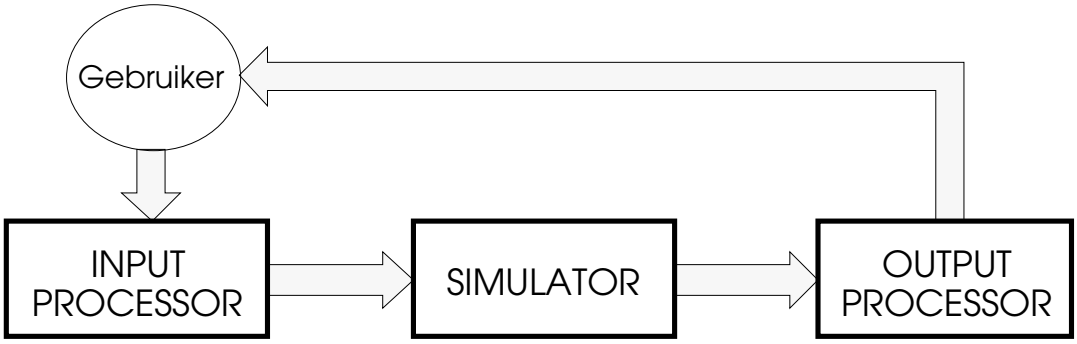
In dit hoofdstuk bespreken we de implementatie van TRIP. Hierbij is gebruik gemaakt van de in de hiervoor gegeven analyse uitgewerkte ontwerpprincipes, zoals deze op de afzonderlijke modules van de software betrekking hebben. Eerst wordt de functie van het programma TRIP nauwkeurig besproken waarna een kort overzicht van de structuur van de software en de samenhang tussen de modules onderling en de gebruiker wordt besproken.

De implementatie van TRIP behelst tot nu toe twee 250-uurs opdrachten van beide auteurs. Waarschijnlijk zouden nu nog ruim twee 250-uurs opdrachten nodig zijn om het programma te completeren. In de vierde fase is zoals eerder beschreven een 'mock-up' van TRIP geschreven die het gedrag van het echte TRIP programma nauwkeurig emuleert. Veel van de complexe algoritmen die oorspronkelijk al geschreven waren voor het genereren van een schaalbaar gebruikersinterface, waaronder de automatische generatie van dialogen, is echter achterwegen gelaten. In plaats daarvan is dit deel nu 'hard-coded', waarbij nadrukkelijk veel aandacht is besteed in aan het voor de gebruiker identiek doen lijken van beide implementaties. De 'mock-up' wordt in het volgende hoofdstuk in meer detail beschreven.

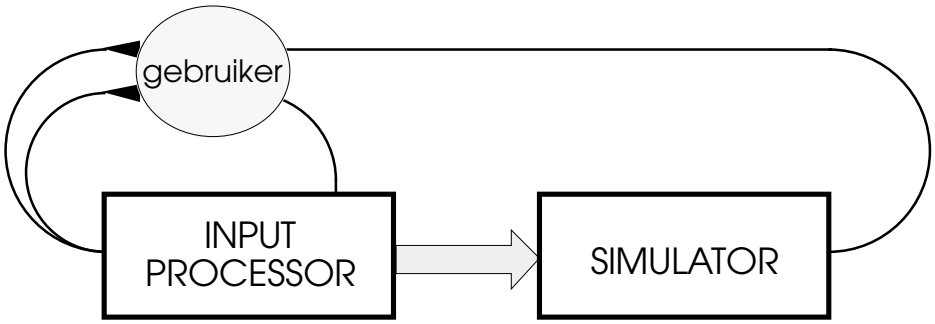
De eerste fase van de ontwikkeling wordt gedetailleerd beschreven in [Middelhoek89]. [Hekster90] beschrijft de tweede fase van de ontwikkeling van TRIP. De derde fase blijft vooralsnog ongeïmplementeerd. We kunnen voor deze fase wat betreft implementatie echter wel een aantal nuttige suggesties doen aan de hand van de eerder gevonden interface-principes, en de ervaringen verkregen tijdens de implementatie van de mock-up die wel een vereenvoudigde versie van de voor deze fase benodigde software bevat.

TRIP

De doelstelling achter het ontwerpen van TRIP is het voor de gebruikers van TRENDY beschikbaar maken van een eenvoudiger te gebruiken, leren en in het algemeen



Figuur 5. Algemeen simulatorsysteem



Figuur 6. Het ontwerpproces

plezieriger gebruikersinterface. Deze eisen leiden tot het ontwerpen van een grafisch gebruikersinterface gebaseerd op directe-manipulatie technieken. Voordat we overgaan tot de beschrijving van dit ontwerp en de implementatie geven we een meer gedetailleerde beschrijving van het probleem.

Een algemeen simulatorsysteem kan voorgesteld worden te bestaan uit drie onderdelen: een input processor, een simulator, en de output processor (figuur 5). De input processor zorgt voor het omzetten van menselijke handelingen in commando's die door de simulator begrepen kunnen worden. Een simulator kan meestal geïnstrueerd worden met behulp van een primitief toepassingsgericht programmeertaaltje. De output processor dient voor de transformatie van veelal numerieke data in voor de gebruiker beter interpreteerbare vormen, zoals grafieken en plots. In de huidige situatie ontbreekt de eerste stap echter, de input processor, helaas vaak.

TRENDY, zoals zoveel state-of-the-art simulatieprogramma's, wordt gebruikt in een *batch*-achtige manier. Dit is het gevolg van de niet geringe hoeveelheid tijd die nodig is om een simulatie door te rekenen. Zelfs wanneer snellere computers beschikbaar komen zal de simulatietijd vermoedelijk niet veel verminderen, aangezien de complexiteit van de problemen die de ingenieurs willen oplossen meegroeien met de rekenkracht van de computer. Deze status-quo is vermoedelijk inherent aan state-of-the-art wetenschap.

Het gebrek aan rekenkracht reduceert de interactiviteit van het ontwerpprocess. Het is belangrijk om in het ontwerpproces meer interactiviteit te introduceren. Dit maakt het onder meer mogelijk om een aantal fouten snel te corrigeren. Een ontwerpproces dat twee cycli kent, die eventueel meerdere keren doorlopen worden, is voorgesteld in figuur 6. De eerste cyclus is de invoer van de probleembeschrijving welke zeer interactief is, en de tweede is de simulatiestap die in veel gevallen niet interactief is.

Veel simulatie programma's, zo ook TRENDY, gebruiken een tekst file waarin het te simuleren probleem als invoer staat beschreven. Deze beschrijvingen kunnen variëren van simpele parameterwaarden tot complexe modelbeschrijvingen. De simulator

definiëert een taal voor zijn invoer. In het algemeen kan de syntax van zo'n taal beschreven worden als†:

<taal> ::= { <commando> <separatie_teken> }

Hier staat `<commando>` voor een reeks van 'keywoorden' (zoals `IF`, `THEN`, `BEGIN`) en special characters als ';' and ':' of parameters. Deze regel definiëert een categorie talen welke een subset is van de LR(1) grammatica's. In het bijzonder voldoet ook de TRENDY invoertaal aan deze syntax.

Om de gebruiker te ondersteunen bij het maken van een invoerfile voor TRENDY moet TRIP een aantal taken van de gebruiker overnemen. Bij voorkeur zouden dit de vervelende, repetitieve, niet-creatieve onderdelen moeten zijn. Doel is om de generatie van de invoer specificatie sneller, beter en plezieriger te laten verlopen.

De eerste vraag die beantwoord moet worden, is wat kan TRIP doen om de bovenstaande doelen te bereiken? In de eerste plaats zou het programma het onnodig moeten maken voor de gebruiker om de syntax van de invoertaal te leren. Het moet dus het gebruik van 'recall' memory verminderen en daarvoor in de plaats gebruik moeten maken van 'recognition'. In de tweede plaats zou TRIP ondersteuning kunnen geven bij het specifieren van de commandoparameters. Deze hebben twee eigenschappen, te weten hun *type*, en hun *waarde*. Het type ligt vast voor de categorie talen waaronder de TRENDY invoertaal valt. Voor de waarde geldt dit niet. Wel bepaalt het type het domein van mogelijke waarden en bovendien is vaak een zinvolle default waarde te geven.

De hierboven beschreven ondersteuning is op een laag niveau. Hiernaast zou echter ook nog op een hoger niveau ondersteuning kunnen plaatsvinden. Gedacht wordt hierbij aan bijvoorbeeld blokstructuren (`BEGIN`…`END` of {…}) in de taal, of aan context-afhankelijke commando's. Hierop zal niet verder worden ingegaan.

Een derde niveau van ondersteuning zou kunnen bestaan uit een kennissysteem dat helpt bij het checken van de specificatie op een semantisch niveau. Dit is echter duidelijk werk voor de toekomst.

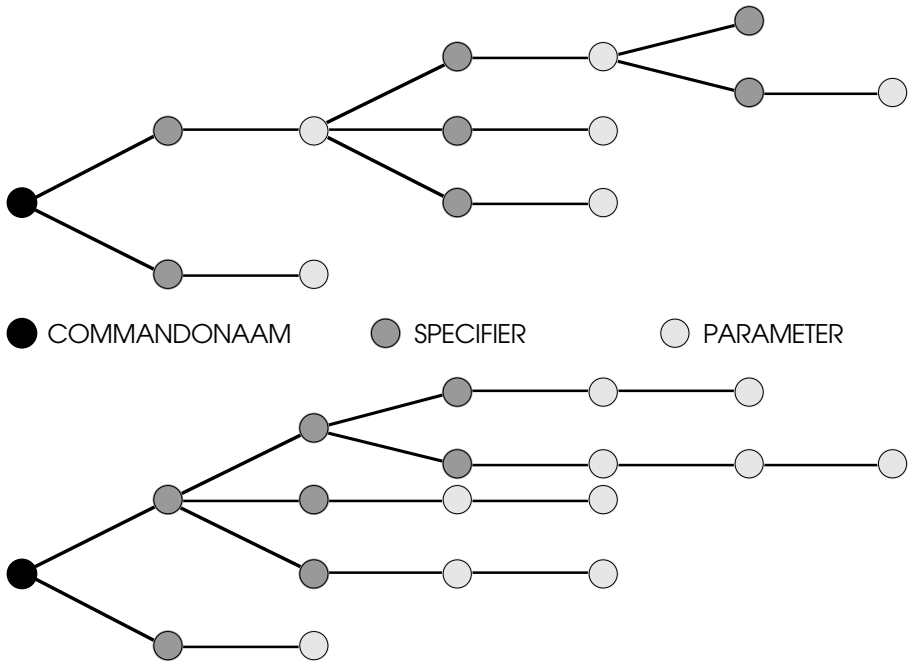
Een belangrijk onderdeel van de eisen is dat TRIP het ontwerpproces moet versnellen. Dit kan op verschillende manieren gebeuren. In de eerste plaats kan het specificeren van commando's versneld worden door ze in pop-up menus te selecteren inplaats van ze in te hoeven typen. Belangrijke snelheidswinst kan verder geboekt worden door het aantal debugsessies te reduceren. Dit kan in TRIP op twee plaatsen gebeuren. In de eerste plaats kan er gedebugged worden op het TRIP-niveau— dat wil zeggen, verbetering van parameterwaarden en voorkoming van intypfouten (bijvoorbeeld door gebruik van menus). Daarnaast blijft er de trage debug-cycle bestaan omdat het nodig is eerst de simulator te starten. Het onderzoek in dit verslag zal hopelijk bevestigen dat door het gebruik van TRIP minder van deze laatstgenoemde cycles nodig zijn.

Belangrijke snelheidswinst is verder te bereiken door het hergebruik van eerder gemaakte specificaties. Het TRIP systeem beschikt daarom over een bibliotheek, of *data base*, die eerder gemaakte specificaties kan bevatten.

Bovendien is er in TRIP getracht de gebruiker een beter overzicht te geven van de specificaties. Dit is onder meer gedaan door iconen te gebruiken die commando's representeren, en onnodige details veroorzaakt door parameters te verbergen tenzij de gebruiker ze expliciet wil zien.

De bovenstaande eisen zijn allemaal direct zichtbaar voor de gebruiker. Er zijn echter andere, minder zichtbare eisen waaraan TRIP moet voldoen. Hieronder vallen de eisen dat TRIP de flexibele en makkelijk uitbreidbare structuur van TRENDY moet kunnen ondersteunen. Dit heeft geleid tot een ontwerp dat bestaat uit een vaste kern met daarom heen een makkelijk veranderbare definitielaag. Een extra voordeel hiervan is dat dit het mogelijk maakt om TRIP te gebruiken voor compleet andere simulatie programma's.

De transformatie van in- naar uitvoer is 'hard-coded', wat de mogelijke grammatica's van de uitvoertalen beperkt. De manier waarop deze transformatie wordt uitgevoerd beperkt de grammatica van de uitvoertaal nog verder. Om te beginnen zullen we de



Figuur 7. Transformatie van de commandosyntax in een vorm die geïmplementeerd kan worden met behulp van menus

algemene syntax van een commando van de invoertaal bekijken. De volgende algemene structuur kan herkend worden:

<commandonaam> { [<specifier>] [<parameter>] } <eindtoken>

Een voorbeeld hiervan in TRIP is:

add material = oxide impurity = boron value = 2.0e15

Deze heeft de volgende structuur, waarin alle nonterminals gereduceerd zijn tot de constitutionele terminalsymbolen:

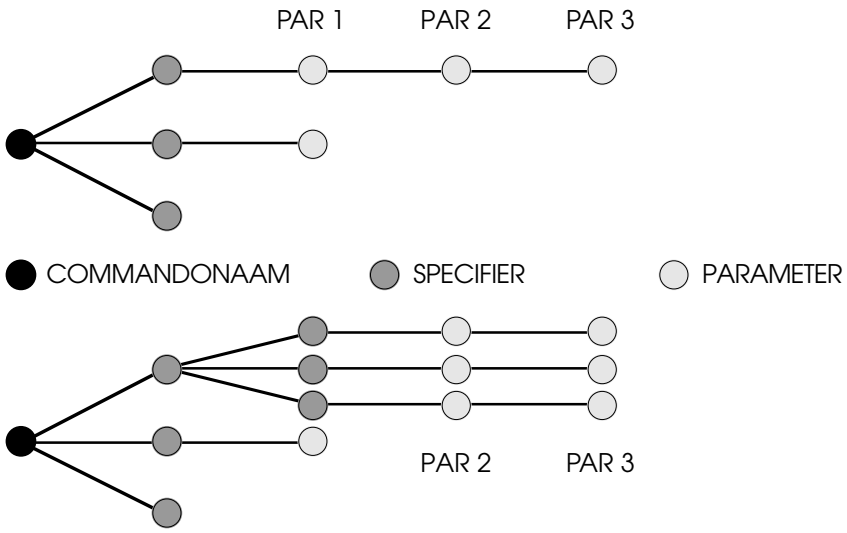
<commandonaam> <specifier> <parameter> <specifier> <parameter> <specifier> <parameter> <eindtoken>

In de meeste gevallen hangen de parameters die een commando nodig heeft af van de gebruikte specifiers. Hierbij is de set van toegestane specifiers normaal beperkt. Dit maakt het mogelijk om deze bijvoorbeeld te selecteren vanaf een menu. Voor de 'ongemode' implementatie met behulp van menus zou het nuttig zijn om de structuur van commando's te transformeren naar de volgende:

<commandonaam> { <specifier> } { <parameter> } <eindtoken>

Onderstaande figuur laat twee boomstructuren zien die de verschillen tussen de twee mogelijke commandostructuren illustreren:

Indien de eerste commandostructuur gebruikt wordt dan bestaat het selecteren van een commando uit het kiezen van een 'commandonaam' uit een menu en het selecteren van de eerste specifier uit een submenu. Hierna moet de eerste parameter worden ingevoerd. De andere parameters kunnen nog niet ingevoerd worden omdat deze afhangen van de nog niet geselecteerde specifiers. In de volgende stap moet de tweede specifier gekozen worden uit een nieuw menu. Dit nieuwe menu maakt het gebruikersinterface modaal en daarom is dit niet wenselijk. Deze structuur zou het onder meer veel moeilijker maken om een eerder gespecificeerde parameterwaarde te veranderen zonder weer van het begin



Figuur 8. Transformatie van parameters naar specifiers

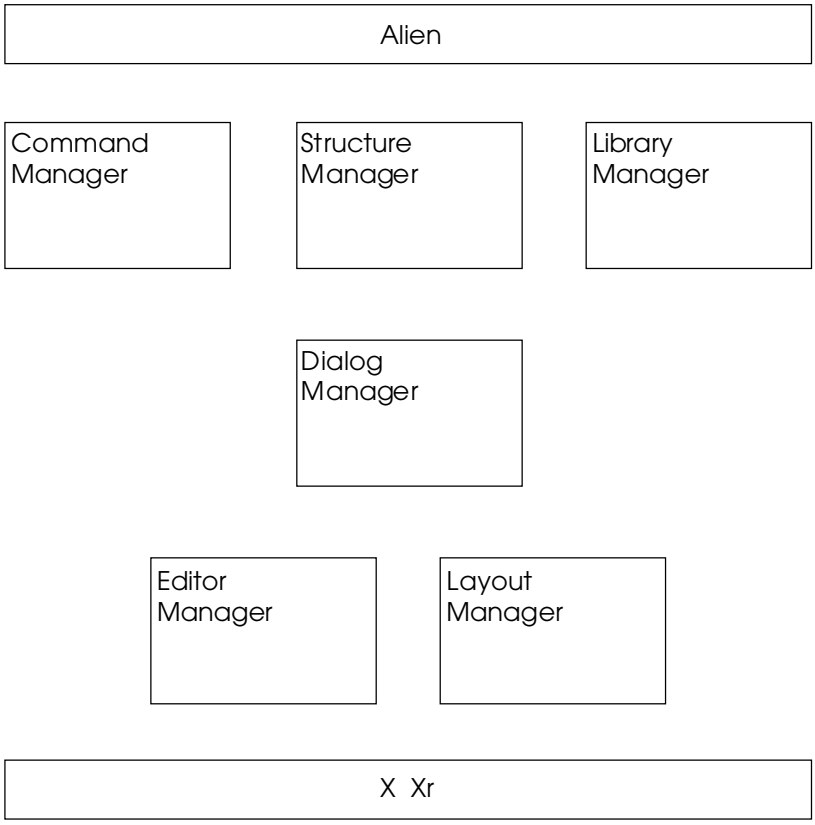
af aan te beginnen. Bovendien zou het verschijnen van een nieuw menu of nieuwe keuzemogelijkheden de esthetische integriteit en stabiliteit van het interface aantasten.

Aan de andere kant laat de tweede structuur toe om een commandonaam en de specifiers in één handeling te selecteren, door middel van een hierarchische selectie van een menu en submenus. Deze methode maakt het niet nodig om menus te veranderen en ook het probleem van het corrigeren van een eerder ingevoerde parameter is niet meer aanwezig aangezien alle parameters op hetzelfde moment veranderd kunnen worden. Het is zelfs mogelijk om het invoeren van commando's helemaal ‘modeless’ te maken door het gelijktijdig selecteren van nieuwe commando's en invoeren van parameters van andere commando's mogelijk te maken.

De vraag is nu of de eerste vorm getransformeerd kan worden in de tweede, en zo ja, aan welke eisen de commandotaal dan moet voldoen om dit mogelijk te maken. Deze transformatie is zonder meer mogelijk zolang de specifiers onafhankelijk zijn van eerdere parameter waarden. Dit wil feitelijk zeggen dat er geen cyclische afhankelijkheid mag bestaan tussen parameters en specifiers. Als deze inderdaad onafhankelijk zijn is de tweede vorm eenvoudigweg een expansie van de eerste. Mocht de afhankelijkheid wel bestaan dan rest er nog een laatste mogelijkheid door de parameter te beschouwen als een specifier, maar dit kan practische problemen opleveren indien de set van waarden die de parameter kan aannemen groot is.

De geëxpandeerde vorm gebruikt meer opslagruimte en bevat bovendien redundante informatie. Het is mogelijk dit te reduceren door de expansie pas op run-time te laten plaatsvinden, maar hier is niet voor gekozen omdat het de toch al vrij grote complexiteit zou doen toenemen. Met name zou de interpreter die de commandodefinities leest aanzienlijk complexer worden. Bovendien is het handmatig expanderen en invoeren niet zoveel extra werk.

Afsluitend willen we nog een transformatie voorstellen tussen twee equivalente beschrijvingen, die het mogelijk maakt om tot op zekere hoogte te kiezen hoeveel niveaus er in de menustructuur komen. Een menuselectie komt in feite neer op de selectie van een waarde uit een beperkte set van mogelijke waarden. Hetzelfde geldt eigenlijk voor het invoeren van een parameter. Dit is eigenlijk ook het selecteren van een waarde uit een set, zij het dat deze set oneindig veel elementen kan bevatten zoals in het geval van reëelwaardige parameters. Deze transformatie is in de volgende figuur geïllustreerd. Waarvoor uiteindelijk gekozen wordt hangt onder meer af van de vereiste symmetrie, snelheidseisen, en geheugengebruik.



Figuur 9. Architectuur van TRIP

Afsluitend nog een aantal opmerkingen met betrekking tot TRIP. De transformaties waarvan in TRIP gebruik is gemaakt vormen de aanzet tot het oplossen van een grotere klasse van probleemen, namelijk het (automatisch) transformeren van commandotalen naar grafisch-geörienteerde gebruikersinterfaces. Met de tot nu toe verworven inzichten in het probleem lijkt dit zeer wel mogelijk, zij het dat het nog zeer veel tijd zou vergen. Programmatuur die dit kan zou van groot nut kunnen zijn voor met name wetenschappelijke software die vaak meer op functionaliteit dan op bruikbaarheid is ontworpen.

OPBOUW VAN TRIP

De interne architectuur van het complete TRIP systeem is in de figuur weergegeven.

Het systeem is van het begin af aan met een stricte modulariteit ontworpen— de belangrijkste functies zijn elk ondergebracht in een onafhankelijke “manager” (beheerder) van deze functie. Dit principe is gebaseerd op het gebruik van zogenaamde *abstracte datatypes* [Amstel88, deel 2, p. 148]. Deze stricte scheiding van verantwoordelijkheden bevordert de mogelijkheden tot uitbreiding of modificaties aan het programma. Ook wordt hierdoor de mogelijkheid voor het implementeren ervan door meerdere programmeurs bevorderd.

In het ontwerp van TRIP is een scheiding aangebracht tussen de functionele en de interface-delen van de programmacode aangezien voor deze laatste nog geen duidelijke standaard beschikbaar was.

Één van de belangrijkste aspecten van een gebruikersinterface is de manier waarop data wordt gerepresenteerd. Binnen TRIP is een onderscheid gemaakt tussen twee niveaus van representatie. Het laagste niveau is de representatie van individuele commando's,

terwijl het hogere niveau zich bezighoudt met de onderlinge samenhang tussen commando's. Beide worden geïmplementeerd in een aparte manager. De Commando Manager is verantwoordelijk voor de selectie van commando's in menus en voor the initiëring van de parameterwaardespecificatie. De onderlinge samenhang van commando's wordt beheerd door de Structure Manager.

Al eerder is gesteld dat TRIP flexibel en uitbreidbaar moet zijn. Dit wordt onder meer bereikt doordat de definitie van het te genereren gebruikersinterface in een aparte "Command Dialog Description File" is ondergebracht. Bovendien is het mogelijk om in een later stadium zonder dat hercompilatie van TRIP nodig is nieuwe methoden voor de invoer van commando's toe te voegen. Hierbij kan bijvoorbeeld gedacht worden aan het toevoegen van software die een thermometer op het scherm afbeeldt waarop door middel van directe manipulatie een temperatuur in te stellen is. Deze laatste mogelijkheid is met name van belang als de software verspreid zou worden of voor andere simulatoren gebruikt zou gaan worden.

Noodgedwongen kunnen een aantal interessante onderwerpen hier niet besproken worden. De geïnterreseerde lezer kan meer vinden in met name [Middelhoek89a] en [Hekster90]. De rest van dit hoofdstuk is gewijd aan beschrijvingen van ieder van deze modules afzonderlijk. Deze bestaan uit een zeer beknopte schets van de plaats van de desbetreffende module binnen het systeem. Elk van deze beschrijvingen wordt gevolgd door een beperkte beschrijving van de principes die betrekking hebben op de gebruikersvriendelijkheidsaspecten. Hierbij is nadrukkelijk aandacht besteed aan het oogpunt van de gebruiker van het programma. We beginnen door een aantal globaal van toepassing zijnde principes te beschrijven.

GLOBALE TOEPASSINGEN

Consistentie. Op het ogenblik omvat de implementatie ongeveer 9000 regels portable C programmacode geschreven onder het UNIX operating systeem. Er wordt uitvoerig gebruik gemaakt van het X Window systeem van MIT [Hall88] en de Xr toolkit van Hewlett-Packard [HP87]. De implementatie draait momenteel op HP 9000 serie 800 workstations maar is door het consequent toepassen van software standaarden hardware-onafhankelijk. Natuurlijk helpt zelfs het gebruik van standaarden niet als een fabrikant stopt met de ondersteuning ervan. Met name wat dit laatste betreft heeft de softwareindustrie nog een verre van voorbeeldige reputatie.

Esthetische integriteit. De gebruiker heeft contrôle over de gebruikte kleuren in het systeem en de lettertypen door middel van specificaties in zijn persoonlijke ".Xdefaults" file. Dit is een standaard X file waarin dit soort preferenties voor alle X applicaties kunnen worden gespecificeerd. Gebruikers zullen op deze manier ook andere parameters van het programma naar eigen voorkeur kunnen aanpassen. Het X Window systeem definieert al de structuur en inhoud van zulke files, zodat ook hier de consistentie tussen verschillende programma's gewaarborgd is.

Minimalisatie geheugengebruik. Door het aanbieden van 'clipboard' functies (knippen, copiëren, plakken) is het in veel mindere mate nodig dat de gebruiker grote hoeveelheden gegevens hoeft te onthouden en reproduceren. Bovendien vereenvoudigt dit het uitwisselen van gegevens tussen TRIP en andere programma's.

COMMAND MANAGER

Alhoewel de Command Manager zelf geen gebruikersinterface in de traditionele zin heeft speelt het wel een hoofdrol in de generatie van het interface. Deze module leest een externe specificatie van het te genereren gebruikersinterface en zet deze om in een interne structuur die door andere delen van het programma gebruikt wordt om het eigenlijke gebruikersinterface te genereren.

Voorkom ingebouwde afhankelijkheden. In TRIP is de commando- en dialoogstructuur in een volledig onafhankelijk bestand (de *Command/Dialog Description file*) gespecificeerd. De CDD file kan door gebruikers van het systeem zelf veranderd worden. Zo is het mogelijk om het TRIP systeem later zeer eenvoudig toepasbaar te maken voor uitgebreide of verbeterde versies van de simulator, of zelfs voor andere, radicaal verschillende simulators. De CDD file bevat dus de specificatie van het door TRIP te genereren gebruikersinterface. Tevens kan deze file gezien worden als een andere kant van het user-interface van TRIP, dat weliswaar slechts door experts gebruikt hoeft te worden. TRIP ontleent veel van zijn flexibiliteit en kracht aan het gebruik van deze constructie.

Belangrijke user-interface aspecten zijn terug te vinden in de Command Manager aangezien deze bepaalt welke gegevens noodzakelijk zijn en in welke vorm om een volledige grafische gebruikersinterface voor een tekst-gestuurd simulatorprogramma te genereren. Het liefst willen we een zo flexibel mogelijke interfacegenerator die met zo min mogelijk initiële informatie een goede gebruikersinterface kan genereren. Helaas zijn deze eisen vaak tegenstrijdig. Om bijvoorbeeld goede dialogen te genereren is het vaak nodig om vrij gedetailleerde informatie over de inhoud ervan te geven. Naast gegevens als hoeveel onderdelen er in een dialoog geplaatst moeten worden en van welk type deze zijn is het vaak wenselijk om extra informatie over de logische samenhang tussen de verschillende delen beschikbaar te hebben. Deze gevens moeten worden opgeslagen in de CDD file. Doel is dus het vinden van een minimale interfacespecificatie-syntax die het mogelijk maakt geheel automatisch een gebruikersinterface, dat aan bepaalde kwaliteitseisen voldoet, te genereren. Deze eisen zijn terug te vinden in de vorm van de eerder beschreven basis principes.

Voor TRIP is gekozen voor een representatie van commando's op twee niveaus, te weten hierarchische menu's en dialogen. Naast informatie die beschrijft hoe deze menus en dialogen in elkaar zitten gebruikt TRIP extra gegevens over de parameters om het mogelijk te maken om vroegtijdig, voor een simulatie wordt uitgevoerd, al een aantal controles uit te voeren zodat fouten reeds nu kunnen worden opgespoord en verbeterd. Dit komt overeen met de kleine ontwerpcyclus uit figuur 6. In de eerste implementatie van TRIP is gekozen voor een syntax die de volgende elementen bevat:

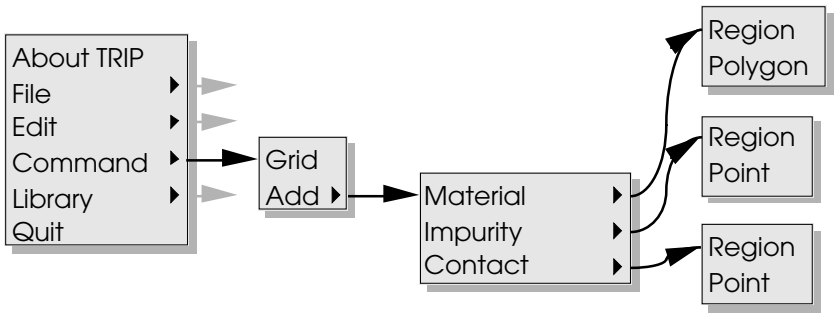
- hierarchische menus
- dialogen
- type definities van dialoog elementen
- specificatie van initiele of 'default' waarden van de dialoog elementen
- hierarchische definitie van data typen
- specificatie van iconen die de commando's kunnen representeren

Tot slot willen we de werking van de Command Manager illustreren met een eenvoudig voorbeeld. Het onderstaande bevat een fragment uit een CDD file die een stuk van het interface voor TRENDY beschrijft.

```
/*
 * This file contains the proposed structure for the TRENDY commands.
 * It is based on the syntax as available at March 3, 1989
 */

/* Now some variable types are defined */
define posreal as real;
lowerbound posreal is 0.0;
define temp as posreal;

/* The command i.e. the menu / dialog structure is now given */
(Add : AddIcon
        (Material
                (Region
                        <material = : 'silicon, oxide, nitride, poly, gas, ambient, back' :
```



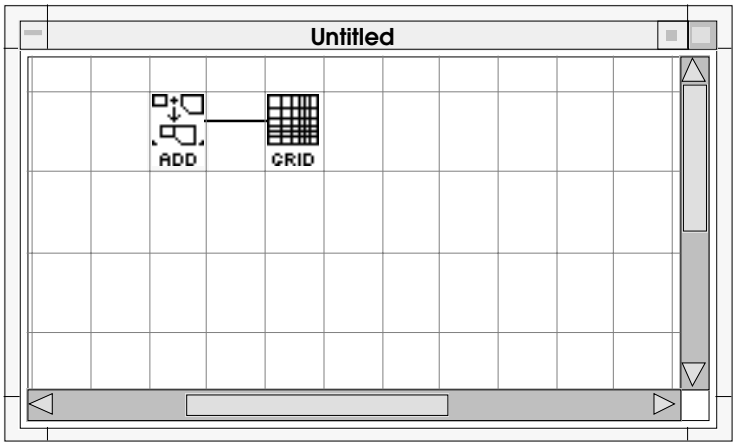
Figuur 10. Voorbeeld van de hierarchische menustructuur zoals die voor TRENDY gegenereerd wordt.

```
                    'silicon'>
                <impurity = : 'arsenic, boron, phosphorus, antimony, potential,
                    electrons, holes, vacancy, interstitial' : 'arsenic'>
                <value = : posreal : 0.0>
                <x.left = : real : 0.0>
                <x.right = : real : 0.0>
                <y.left = : real : 0.0>
                <y.right = : real : 0.0>
            )
            …
            …
            (Polygon
                < material : 'silicon, oxide, nitride, poly, gas, ambient, back' :
                    'silicon'>
                <impurity = : 'arsenic, boron, phosphorus, antimony, potential,
                    electrons, holes, vacancy, interstitial' : 'arsenic'>
                <value = : posreal : 0.0>
                {<x = : real : 0.0>
                <y = : real : 0.0>}
            )
        )
        (Impurity
            (Region
                …
            )
            (Point
                …
            )
        )
        (Contact
            (Region
                …
            )
            (Point
                …
            )
        )
);
```

In het bovenstaande voorbeeld zijn duidelijk de definities van commandos en die van data typen te herkennen. Uit de commando definities kan op eenvoudige wijze de hierarchische menustructuur worden afgeleid. Deze is gedeeltelijk weergegeven in figuur 10.

STRUCTURE MANAGER

Deze manager, die deel uitmaakt van de fase III van TRIP, is vooralsnog ongeïmplementeerd. Omdat we niettemin toch erg graag de functie van het programma op



Figuur 11. Window van de Structure Manager dat de iconen die twee commando's representeren bevat

commandostructuur-niveau wilden testen hebben we een tijdelijk ‘surrogaat’ ontworpen en geïmplementeerd. Dit is echter slechts een minimum implementatie. Te verwachten is dan ook dat de uiteindelijke versie aanzienlijk betere resultaten zal opleveren.

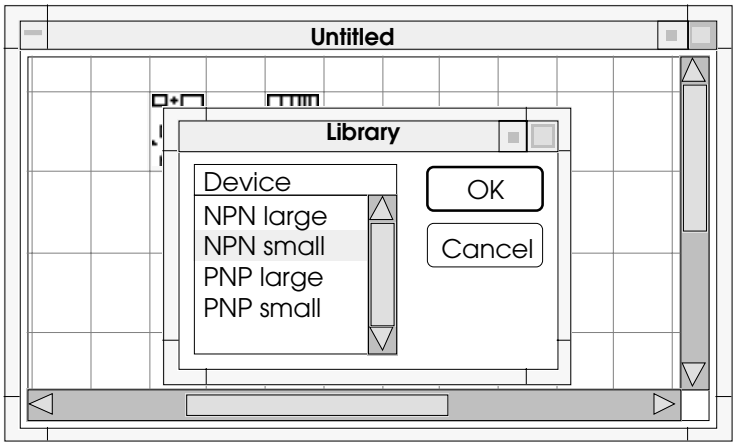
Het metafoorprincipe vindt in de Structure Manager een zeer belangrijke toepassing: In plaats van het leren van een complete specificatietaal denken TRIP-gebruikers alleen in termen van iconen, relaties tussen iconen en dialogen die commando's representeren. Zodoende worden ze helemaal van de specificatiesyntax afgeschermd zodat deze alleen gebruikt wordt als communicatie tussen TRIP en de simulator. Doordat deze syntax niet meer geleerd hoeft te worden wordt ondermeer een reductie van het gebruik van recall geheugen bereikt.

De commando's van de taal kunnen uit menu's worden geselecteerd. Alhoewel dit onder de oorspronkelijke Xr-gebaseerde versie niet mogelijk is, hebben we in de mock-up op de Macintosh in de menu's de namen van de commando's samen met de desbetreffende iconen gepresenteerd, om de associatie tussen beide te versterken. Alhoewel we in het algemeen geadviseerd hebben tegen het gebruik van meedere niveau's diepte van hierarchische menu's, achtten we dat in ons geval echter wel wenselijk, zoniet noodzakelijk. De menustructuur reflecteert nu precies de hierarchie in de commandostructuur, en ook al kan dit drie niveau's diep zijn, vinden wij dat een ‘splitsing’ middenin deze hierarchie alleen verstorend zou werken.

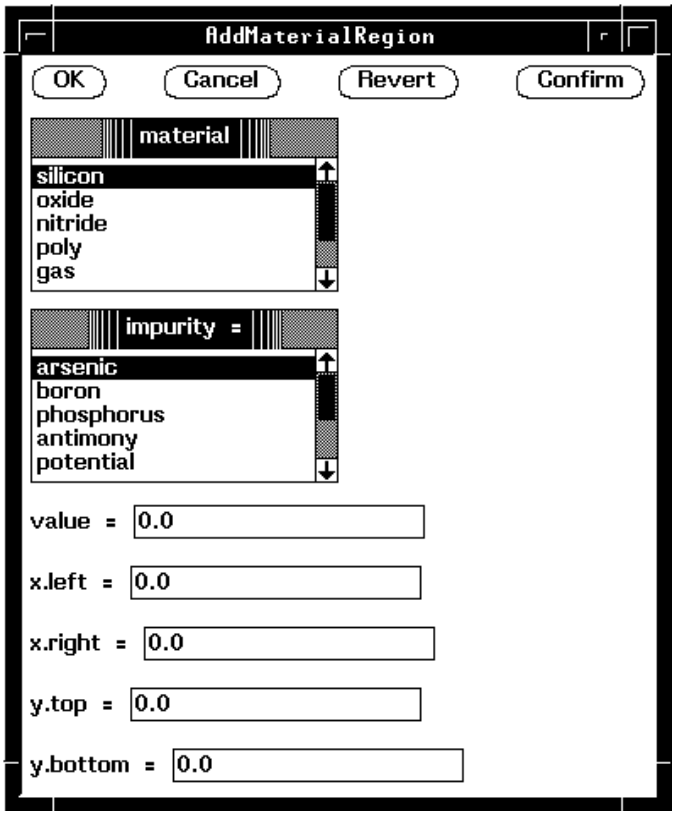
De interactie vindt plaats met behulp van directe-manipulatie technieken. Zo kan bijvoorbeeld de tijdsvolgorde waarin simulatiestappen doorlopen worden met behulp van het tekenen van een pijl tussen de stappen aangegeven worden. Het zal duidelijk zijn dat de Structure Manager uitgebreid gebruik maakt van de metafoor, directe manipulatie en van de iconische representatie principes.

LIBRARY MANAGER

De Library Manager is bedoeld om veel gebruikte commandospecificaties en groepen van commandospecificaties te bewaren en snel weer opvraagbaar te maken. Één van de basisprincipes is hier dus hergebruik. Naast de directe tijdbesparing en het voorkomen van fouten is er een derde groot voordeel aan het hergebruik van commando's: onervaren gebruikers kunnen bibliotheken van ervaren gebruikers gebruiken en deze desgewenst aanpassen aan de eigen wensen, zonder dat ze zelf alle details van de simulator hoeven te weten. Deze situatie treedt met name vaak op in universiteiten, waar studenten vaak in korte tijd een aantal experimenten moeten uitvoeren. De tijd laat het vaak niet toe om



Figuur 12. Selectie van een element, in dit geval een set van commando's, uit een bibliotheek

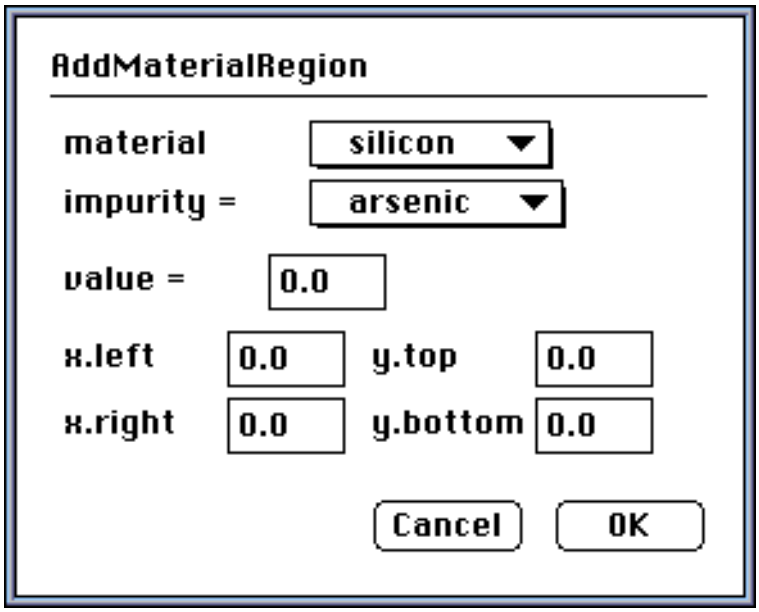


Figuur 13. Voorbeeld van een dialoog van het TRENDY-commando `Add-Material-Region`

voor een practicum een compleet programma te leren gebruiken en dit is vaak ook eigenlijk niet wenselijk.

DIALOG MANAGER

Deze module verzorgt het presenteren van *dialogen* aan de gebruiker. Een dialoog is een voor één specifiek commando of een groep van functioneel en structureel gerelateerde



Figuur 14. Gebruik van pop-up menus in plaats van lijsten in dialogen

commando's geoptimaliseerde *window* waarin de parameters van het commando door de gebruiker kunnen worden gespecificeerd. Een voorbeeld van een dialoog zoals dat door TRIP voor het TRENDY Add-Material-Region commando gegenereerd kan worden is in figuur 13 gegeven.

In het oorspronkelijke werk is ook gekeken naar het invoeren van arrays van waarden, of in het algemeen, van metawaarden, waarvan arrays een speciaal geval zijn. Naar de effectiviteit van de voorgestelde invoermethoden van metawaarden is hier geen onderzoek verricht.

In de TRIP mock-up maken we in dialogen gebruik van pop-up menus in plaats van de list editors van de eigenlijke TRIP software. Pop-up menus waren in de oorspronkelijke toolkit (Xr intrinsics) niet beschikbaar maar vormen een beter alternatief voor de weinig subtiele opsomming van alle mogelijke opties van alle parameters. Bovendien nemen Pop-up menus in dialogen minder ruimte in beslag dan een list editor doordat ze in eerste instantie slechts de relevante huidige waarde laten zien, en hebben verder alle voordelen van gewone menus: de alternatieven zijn met één enkele muisklik zichtbaar en kunnen uiterst eenvoudig en snel geselecteerd worden. Bovendien wordt de mogelijkheid tot generatie van visueel aantrekkelijke dialooglayouts aanzijnlijk verbeterd doordat de grote van de dialoogelementen uniformer is.

Dat pop-up menus inderdaad een beter aanzicht geven moge duidelijk worden uit een mogelijke heruitvoering (figuur 14) van bovenstaande dialoog (figuur 13) onder Xr naar de mock-up omgeving op de Macintosh.

Stabiliteit. Tijdens het opbouwen van de dialogen veranderen de grootte van de benodigde schermoppervlak en beschikbare schermlocaties aanzienlijk. Om deze reden houden we tijdens het dialoogopbouw-proces de windows onzichtbaar. Pas wanneer het proces voltooid is wordt de window en zijn inhoud in één keer zichtbaar gemaakt.

Bij het tekenen van de dialogen letten we er op dat alleen het noodzakelijke wordt getekend. Voor sommige programmeurs blijkt het te verleidelijk om bijvoorbeeld altijd de hele dialoog opnieuw te tekenen. Dit heeft echter een bijzonder irritant knipperen van de dialoog tot gevolg. Hoewel aan dit aspect veel tijd en aandacht is besteed, maakt de gebruikte Xr toolkit het consequent toepassen ervan vrijwel onmogelijk.

We gebruiken zover mogelijk standaard terminologie binnenin de dialoog— een “OK” button geeft aan dat de gemaakte veranderingen correct zijn en door het systeem gebruikt moeten worden, en sluit de dialoog. “Cancel” maakt dat de veranderingen vergeten worden en sluit de dialoog.

Hiernaast introduceren we een tweetal gerelateerde functies die door buttons geactiveerd worden: “Revert” herstelt de dialoog, nadat door de gebruiker al eventuele veranderingen zijn gemaakt, tot zijn oorspronkelijke staat. “Confirm” merkt de gemaakte veranderingen als correct en maakt ze bekend aan het systeem, maar houdt de dialoog nog op het scherm voor verdere veranderingen. Merk op dat de functies van “Confirm” en “Revert” gelijk zijn aan die van “OK” en “Cancel”, behalve dat ze de dialoog niet verwijderen. Ook hier is sprake van een symmetrie in het ontwerp.

Esthetische integriteit. Op het eerste gezicht lijkt het misschien een goed idee om deze buttons een kleur te geven: “OK” en “Confirm” zijn ‘gevaarlijke’ functies omdat ze gegevens veranderen, en zouden daarom rood gekleurd kunnen worden. “Cancel” en “Revert” zijn daarentegen ‘veilig’ omdat . Echter heeft de kleur rood een aandacht-trekkend effect zodat deze keuze volkomen averechts kan werken.

Modaliteit. Dit is hierboven al in een andere context genoemd. [Apple85 pp. I-67] behandelt in meer detail het verschil en de toepasbaarheid van ‘modale’ en ‘niet-modale’ dialogen. Een modale dialoog moet eerst volledig afgehandeld worden voordat de gebruiker kan verder gaan met een andere actie, terwijl een niet-modale dialoog op een voor de gebruiker geschikte tijd afgehandeld kan worden.

Dialogen worden vaak gepresenteerd om simulatorcommando's en de bijbehorende parameters te kunnen specificeren. Dit is niet een inherent modale bezigheid. Het is bijvoorbeeld erg goed mogelijk dat de gebruiker tegelijk meerdere commando's wil kunnen zien of veranderen. In onze mock-up is echter afgezien van het gebruik van niet-modale dialogen aangezien dit naar onze inschatting teveel extra programmeerwerk zou vereisen en voor het uitvoeren van de tests niet-essentiëel is.

Een andere veel voorkomende vorm van dialogen is bij het optreden van fouten. In dit geval is de taak van de dialoog om de gebruiker van de aard van de opgetreedde fout te informeren en te wachten totdat de gebruiker zich hiervan op de hoogte heeft gesteld. Dit soort dialogen wordt in de terminologie van [Apple85] een **alert** genoemd. Het is duidelijk wel een modale vorm van communicatie. Het zou namelijk vrijwel zinloos en desoriënterend zijn als de gebruiker willekeurig opgetreden geheugenallocatie of disk I/O-fouten kon negeren en ‘opsparen’, om ze aan het eind van zijn taak allemaal in één keer ‘af te handelen’ (wat dat dan ook zou mogen betekenen).

Een effectieve en interessante presentatie van fouten, waarbij de gebruiker-computer dialoog nog sterker tot zijn recht komt, wordt behandeld in [Efe87] en is de moeite van het implementeren zeer zeker waard.

User-control. Het gebruik van werkwoorden in buttons en menus versterkt het gevoel van de gebruiker dat hij zelf ‘in control’ is. Een voorbeeld is “Cancel” in dialoogbuttons, en alhoewel “OK” strict genomen geen werkwoord is wordt het in de Engelse taal wel zo gebruikt. Dialogen met buttons “Good” en “Bad” met dezelfde functie geven niet het gevoel dat ze een actie induceren.

EDITOR MANAGER

De Editor Manager is dat deel van de software dat zorgt draagt voor het instantiëren van editors binnenin een dialoog. Er bestaan vijf soorten voorgedefiniëerde editors (**primitive editors**) die in de Editor Manager zijn ingebouwd: een *integer* editor voor het specifieren van een interval van integer waarden; een *real* editor voor floating-point getallen; een *string* editor voor het invoeren van karakterstrings; een *Boolean* editor voor het specificeren van Booleaanse (ja/nee) keuzes; en, een *list* editor voor het maken van keuzes uit een lijst van gegeven opties.

De soorten primitive editors zijn zó gekozen dat vrijwel alle dialogen met behulp ervan gegenereerd kunnen worden. Dit maakt het mogelijk om zeer snel nieuwe versies van een simulator of zelfs geheel nieuwe simulatoren in een minimum van tijd te kunnen gebruiken. Zo is het in het bovenstaande deel over de Dialog Manager gepresenteerde TRENDY Add-Material-Region dialog volkomen met behulp van primitive editors opgebouwd.

Soms zal het echter wenselijk of misschien zelfs noodzakelijk zijn om een ander soort editor te kunnen gebruiken. De Editor Manager ondersteunt het definiëren van geheel nieuwe **externe editors** die door andere programmeurs kunnen worden geschreven en op run-time door TRIP kunnen worden gebruikt. Deze editors worden door het programma als externe processen opgestart en aan het hoofdproces gelinkt. Het is zelfs mogelijk om een heel dialoog door een zeer taak-specifieke editor te vervangen en de primitive editors in z'n geheel te vermijden. Door het toevoegen van nieuwe editors zeer eenvoudig te maken (de TRIP software hoeft er zelfs niet voor te worden gehercompileerd) wordt getracht om de ontwikkeling van dit soort gespecialiseerde editors te bevorderen.

Een andere toepassing van het metafoorprincipe: in het geval dat bijvoorbeeld temperatuurwaarden gespecificeerd moeten worden, kan gebruik worden gemaakt van een gespecialiseerde editor in de vorm van een thermometer. Het is echter niet noodzakelijk dat de gebruikte metaforen direct corresponderen met elementen uit de fysische wereld—de verbindingen tussen commando's hebben misschien geen direct fysisch analogon maar zijn wel direct begrijpelijk voor niet-gebruikers.

LAYOUT MANAGER

Tenslotte draagt de Layout Manager zorg voor een esthetisch optimale verdeling van 'zichtbare' elementen binnen een dialoog—zo'n verdeling van elementen heet een **layout** van die elementen. Deze elementen zijn hoofdzakelijk de eerdergenoemde editors maar ook de dialoog *buttons* zoals "OK" en "Cancel" of lege ruimte.

De Layout Manager gebruikt informatie over de logische samenhang tussen elementen om gerelateerde elementen in de dialoog bij elkaar te houden. Er kan een hierarchie van dit soort groepen elementen worden opgebouwd. Het algoritme tracht om de losse en gegroepeerde elementen op een zo esthetisch aantrekkelijk mogelijke manier te verdelen.

Wat wordt er verstaan onder "esthetisch verantwoord"? Om zo weinig mogelijk preconcepties van de programmeur in de software in te bouwen is gekozen voor een zeer flexibele toepassing die op een willekeurig moment op eenvoudige wijze aangepast kan worden. De Layout Manager bouwt de layout constructief op met behulp van **regions**. Dit zijn datastructuren die een eventueel disjunct twee-dimensionaal gebied (strikt genomen deelverzamelingen van $\mathbb{Z}\times\mathbb{Z}$) representeren. Terwijl de layout opgebouwd wordt houdt de Layout Manager het door de dialoogelementen in beslag genomen schermgebied in een region bij. De region constructie functies zijn in staat om met nieuwe elementen bestaande gaten in een region op te vullen. Een extern aangeleverde *region evaluation function* kent aan elk region een 'esthetische kwaliteitswaarde' toe. Door op run-time deze evaluatiefunctie aan te roepen kan de Layout Manager uit een aantal intern gegenereerde alternatieven de beste keuze nemen.

De enige region evaluation die voorlopig door het systeem wordt gebruikt kent een hoge kwaliteitswaarde toe aan regions die een klein oppervlak hebben (omdat compacte dialogen aantrekkelijker zijn dan dialogen met gaten erin) en die een kleine omtrek hebben (om een zo gelijkmatige lengte-breedteverhouding (*aspect ratio*) te prefereren). Hierbij zei opgemerkt dat ergonomisch gezien een 16:9 verhouding waarschijnlijk beter zou zijn, maar indertijd hadden we ook verwacht dat dit later eenvoudig te kunnen veranderen.

OPMERKINGEN

Een belangrijk en voor TRIP zeker ook zeer nuttig onderzoeksgebied is het ontwerpen en implementeren van taak-specifieke editors. Deze kunnen toegepast worden om tot een meer intuïtieve manier van invoeren van parameters te komen. Hierbij komen met name de principes van: 'directe manipulatie', 'gebruik van metaforen', en 'feedback' naar voren. Omdat dit een interessant doch zeer tijdrovend onderzoeksveld is hebben we ons bij de implementatie geconcentreerd op de uitbreidbaarheid van de architectuur, in plaats van te proberen te anticiperen op elke mogelijke toepassing en hiervoor een specifieke editor te schrijven. Dit maakt het extra teleurstellend dat nu we eindelijk op het punt waren gekomen dat we dit hadden moeten kunnen doen we in feite weer van voren af aan moeten beginnen.

Een ander punt waaraan we in het ontwerp van TRIP veel aandacht hadden besteed, met name met het oog op een latere evaluatie, was aan de layout van de gegenereerde dialogen. De Dialog Manager maakt gebruik van vrij geavanceerde algoritmen die de automatische generatie esthetisch aantrekkelijk zouden moeten maken. Voor de herimplementatie en evaluatie hiervan hadden we nu echter geen tijd.

Emulator

Zoals gezegd hadden we te kampen met het vrij ernstige probleem dat we voor deze opdracht het gebruikersinterface moesten testen van een niet-bestaande implementatie. Toen we besloten dat de kans dat de oorspronkelijke TRIP software ooit nog voltooid zou worden nihil was, zijn we naar andere mogelijkheden gaan zoeken om deze opdracht te kunnen uitvoeren.

Ten eerste keken we naar de mogelijkheid om zelf de TRIP software in voldoende mate uit te breiden dat een evaluatie uitgevoerd zou kunnen worden. Dit heeft tot duidelijk voordeel dat we gebruik zouden kunnen maken van een groot deel al bestaande software (ongeveer 9600 regels C code) welke, aangezien we ze zelf hadden geschreven en bovendien goed gestructureerd en gedocumenteerd, we snel effectief zouden kunnen benutten. Helaas bleek dat men bij de vakgroep ICE inmiddels overgegaan was op een geheel nieuwe grafische toolkit (OSF/Motif), zonder dat men uit compatibiliteitsoverwegingen de voorgaande uitvoering (Xr Intrinsics) beschikbaar had gelaten. Dit uiterst ongelukkige voorval maakt dat een zeer aanzienlijk deel van de Fase II TRIP software nu onbruikbaar is geworden.

Er bestond dus een mogelijkheid om ten eerste de bestaande TRIP software om te schrijven van de Xr naar de OSF/Motif Toolkit, en vervolgens de ontbrekende Fase III met behulp van OSF te implementeren. Hier bleken echter een aantal aanzienlijke bezwaren tegen— ten eerste waren wij beide volkomen onbekend met deze toolkit, zodat het dus te veel tijd zou hebben gekost om met deze toolkit te leren programmeren en de (her-)implementatie te verzorgen. Ten tweede bleek het niet eenvoudig om gebruik te maken van de faciliteiten van ICE, omdat men slechts drie van de desbetreffende Hewlett/Packard workstations beschikbaar had die over het algemeen zeer druk bezet zijn.

Er bleef dus weinig anders over dan volledig af te zien van het gebruik van de bestaande TRIP software, en een voor deze opdracht volledig nieuw programma, een emulator, te schrijven. Dit had wel als duidelijke voordelen dat wij beide in het bezit waren van de benodigde hardware en dat wij zeer goed vertrouwd waren met de Macintosh

programmeeromgeving. Uiteindelijk zijn we gekomen tot het schrijven van een TRIP gebruikersinterfaceemulatie bestaande uit ruim 9,000 regels C++ code. Het is misschien meer dan toevallig dat dit aantal zo goed overeenkomt met het origineel.

We hebben hierbij gepoogd de meeste van de essentiële delen van het TRIP gebruikersinterface in de emulator op te nemen. Dit omvat onder andere de Command Manager uit Fase I, de Dialog Manager uit Fase II, en de Structure Manager uit Fase III. We hebben voor zover dit praktisch was de eigenlijke architectuur van de oorspronkelijke TRIP software in de emulator overgenomen, om een zo realistisch mogelijke evaluatie te waarborgen.

Omdat we nu dus in principe de gehele implementatie gebruikersinterface opnieuw in onze eigen handen hadden was het mogelijk om andere dingen te testen dan het geval was geweest wanneer we gebonden waren aan de eigenlijke TRIP software. Bovendien hebben we gezocht naar manieren om deze mogelijkheden zo effectief mogelijk uit te kunnen buiten. Zo was bijvoorbeeld TRIP zelf bestemd voor mensen die op zijn minst enige computer ervaring hebben— bij het ontwikkelen van de emulator hebben we het domein van gebruikers uitgebreid om zelfs bijna volkomen computerleken te omvatten. We kunnen op deze manier nog beter de beweegredenen achter de ontwikkelingen van TRIP testen, namelijk het uitbreiden van het toepassingsgebied van simulatiesoftware.

PRINCIPE VAN DE EMULATOR

De emulator bestaat op een hoog niveau uit een aantal modules die de structuur van de eigenlijke TRIP software zoveel mogelijk proberen na te bootsen. We gaan zodoende dus ook uit van een tekst-gebaseerde simulator waarvoor programma's geschreven moeten worden. Omdat we zogezegd ook graag het programma door niet-experts wilden laten evalueren hebben we een nieuwe taal ontwikkeld die niet in het wetenschappelijke domein ligt. Dit is de “Turtle” taal, gebaseerd op het ‘turtle’ concept van de educatieve LOGO programmeertaal. We achtten dit een goede keuze omdat voor de meeste personen een dergelijke abstractie voldoende dicht bij de dagelijkse praktijk staat. Bovendien heeft het als kenmerk dat zo'n taal structurele overeenkomsten toont met typische simulatortalen. (In principe is een abstractie zoals een “turtle” natuurlijk ook een simulatie van de werkelijkheid).

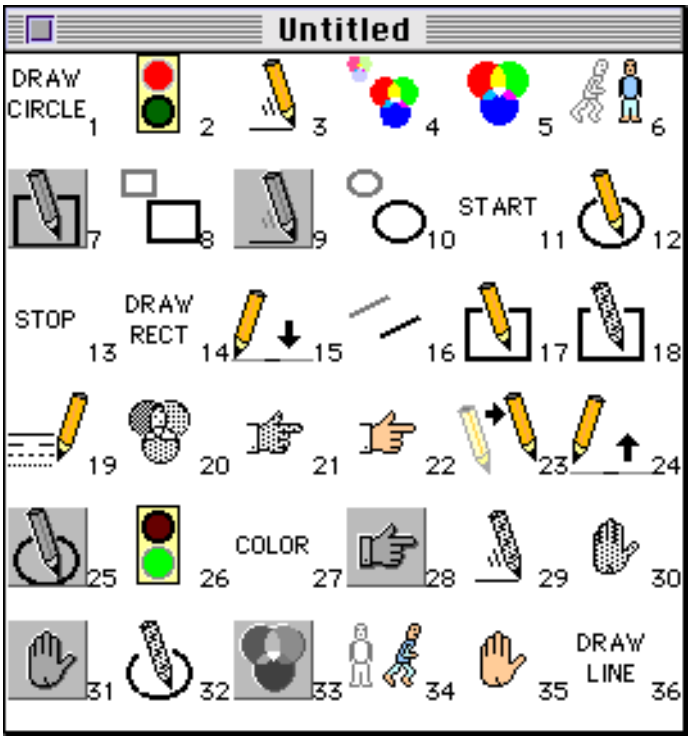
Een essentiëel onderdeel is zodoende ook de Turtle ‘simulator’ zelf. Deze is opgebouwd uit een *lexical analyzer* en *parser* geschreven met behulp van de bekende en in het public domain beschikbare `lex` en `yacc` compilergeneratie software† die we voor deze toepassing naar de Macintosh geport hebben. We hebben een taal met een syntax en grammatica ontworpen (de `lex` en `yacc` specificaties zijn in een appendix gegeven) waarvan we verwachten dat deze conceptueel niet moeilijk is. Hierin zijn we immers bij het testen niet geïnteresseerd, maar juist in het leren gebruiken van de taal. Hierbij zij opgemerkt dat het reduceren van de conceptuele complexiteit van simulatortalen in bepaalde gevallen geen overbodige luxe zou zijn. De taal moet echter ook niet te simpel zijn, aangezien er anders geen leereffect zou zijn om te onderzoeken. Daarvoor is het nodig dat gebruikers alle stadia van onervaren tot ervaren doorlopen.

Op deze manier hopen we bijvoorbeeld effecten te induceren en evalueren waarbij het grafische interface een belemmering kan vormen voor ervaren gebruikers. Merk op dat in de praktijk zo'n situatie zich niet snel zal voordoen omdat de complexiteit van simulatortalen over het algemeen zodanig groot is dat zo goed als alle gebruikers waarschijnlijk nooit het niveau van ervaren ‘uitgeleerde’ gebruiker zullen bereiken, en dus allen gebaat zouden zijn bij een vorm van grafische ondersteuning. Wel levert dit interessante onderzoeksresultaten op.

We wilden ook van de mogelijkheid gebruik maken om wat meer gedetailleerde zaken nader te onderzoeken. Zo speelt het gebruik van iconen in het Fase III deel van TRIP een

---

† eigenlijk GNU `flex` en `bison`



Figuur 15. Icoon herkenningstest

zeer belangrijke rol waarmee wij beide nog geen directe ervaring hadden kunnen opdoen. In het onderzoek is dus een deel opgenomen dat specifiek de rol behandelt van iconenassociaties in het herkennen en herinneren van commando's. Bovendien kon op deze manier de gebruiker alvast voorbereid worden op het zwaardere werk dat op de iconenevaluatie volgt.

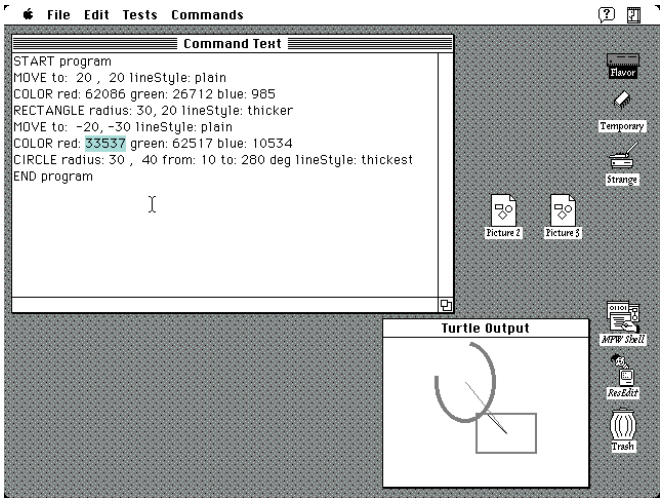
Ook is de mogelijkheid onderzocht om genoeg functionaliteit aan het Structure Manager surrogaat toe te voegen om ook van de Library Manager de functionaliteit te kunnen evalueren, maar dit bleek niet een, binnen de ons tot beschikking staande tijd, reële mogelijkheid.

DE TESTS

Het computer-ondersteunde deel van de gebruikersevaluatie bestaat uit vier delen. We zullen hier in het kort laten zien wat deze inhouden. Terwijl de gebruiker zelf de afzonderlijke tests aan het uitvoeren is, kunnen de resultaten in de tegelijkertijd in de achtergrond in een voor andere programma's bruikbare vorm beschikbaar worden gemaakt. Zo is het bijvoorbeeld mogelijk om vanaf een andere Macintosh die via het netwerk met de 'testmachine' verbonden is in real-time de voorlopige resultaten te zien en te verwerken, zonder dat de gebruiker hierdoor gestoord wordt. We maken voor de verwerking van de gegevens gebruik van een afzonderlijke spreadsheet applicatie waarmee de resultaten automatisch bewerkt en met behulp van grafieken geëvalueerd kunnen worden. Deze gegevens kunnen op hun beurt weer automatisch in het verslag worden opgenomen. Op deze manier hopen we een snelle en nauwkeurige verwerking van de meetresultaten te bereiken.

Icoon Herkenningstest

De tests worden voorafgegaan door een beknopte introductie. In zijn essentie leggen we uit wat de functie is van de turtle taal en wat het doel is dat we ermee proberen te



Figuur 16. Voorbeeld van de tekst programmeertest

bereiken. Vervolgens presenteren we een palet van 36 commando-iconen (figuur 15). De iconen bestaan uit 6 groepen van 6 commando's, maar worden (in tegenstelling tot de figuur) vóór het begin van de test door de computer willekeurig door elkaar geschoven zodat niet uit de plaats van de iconen af te leiden is welk commandoicoon in één groep correspondeert met dat uit een andere groep. In de later te volgen grafische programmeertests wordt één groep van deze iconen gebuikt. We evalueren in hoeverre de gebruiker alleen aan de hand van de iconen zelf en de wetenschap dat het om commando's voor een tekenprogramma gaat de functie van de diverse iconen af te leiden.

Het doel van deze test is beter inzicht te verkrijgen in de optimale vormgeving van iconen. Hierbij kan gedacht worden aan bijvoorbeeld kleur versus zwart/wit, maar ook aan de in het hoofdstuk “principes van user-interfaces” beschreven drie manieren van iconische representie; commando, object–functie en voor/na. De evaluatie vindt plaats aan de hand van de schriftelijke en mondelinge opmerkingen van de gebruiker.

Icoon selectietest

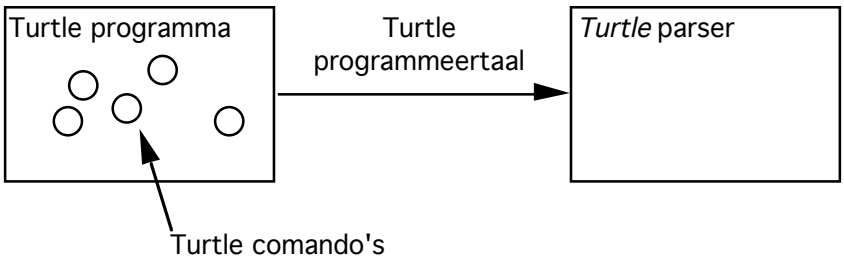
In deze test krijgt de gebruiker achtereenvolgens één voor één commandobeschrijvingen op het scherm gepresenteerd. Nadat voldoende tijd is geven om deze te lezen wordt een palet met iconen gepresenteerd. Doel is nu om de bij de beschrijving behorende commandoicoon in het palet te localiseren en selecteren. We doen hierbij reactiesnelheidmetingen, maar de gebruiker wordt om onnodige druk te voorkomen hierover niet ingelicht. De gebruiker wordt slechts gevraagd de juiste icoon aan te klikken en in geval van een foutieve keuze opnieuw te proberen.

Een gunstig neveneffect van het doen van deze twee eerste tests is dat de gebruiker meteen nuttige informatie leert die van pas komt bij het uitvoeren van de volgende twee opdrachten, dat wil zeggen de mogelijke commando's en in het geval van de grafische programmeertest ook de iconen.

Tekst programmeertest

De gebruiker moet in deze test Turtle programma's schrijven. De test wordt uitgevoerd in een omgeving zoals die wordt weergegeven in de volgende figuur 16.

In een window (“Command Text”) kan de invoer voor het ‘simulatieprogramma’ ingevoerd worden. De cursor neemt wanneer deze zich binnenin de invoer window bevindt de vorm aan van een “I-beam”. Dit is volgens de Apple user interface beginselen vereist, maar heeft niet louter een *consistentie* functie voor ervaren Macintosh gebruikers, maar tevens een *suggestieve* werking die ook voor eerste-keer gebruikers nuttig kan zijn. Het



Figuur 17. Schematische voorstelling van het programmeerproces

veranderen van de cursor geeft aan dat er iets bijzonders aan de hand is en vormt daarmee dus een welkome ondersteuning voor de gebruiker.

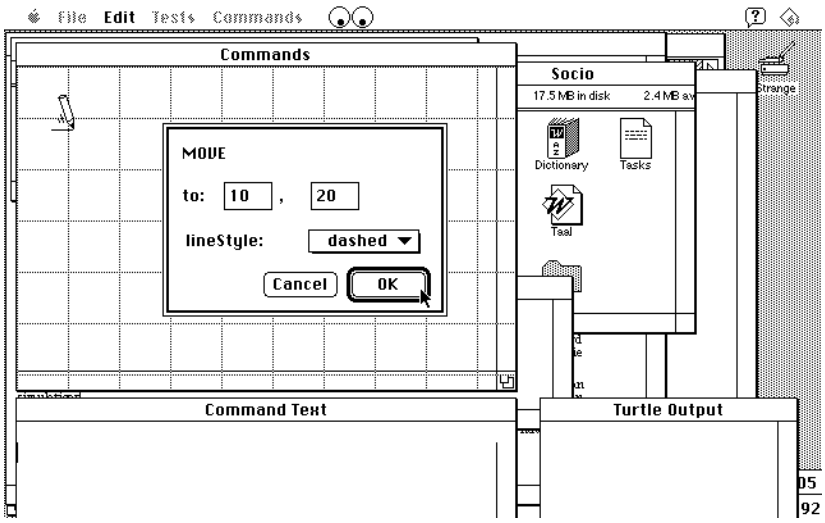
In één van de menus is een “Compile” commando opgenomen waarmee het ingevoerde programma gecompileerd kan worden. De uitvoer van de simulator wordt in een apart window (“Turtle Output”) gepresenteerd om de separatie tussen in- en uitvoer te benadrukken. Om de ‘irritatiefactor’ van de werkelijke situatie na te bootsen gaat er een vertraging aan het compilatieproces vooraf, ter simulatie van de vertraging die optreedt als een echte simulator een complex probleem doorrekent. (Deze vertraging is er de oorzaak van dat het eerder beschreven twee-traps ontwerpproces aan interactiviteit verliest.) Wanneer er een syntaxfout in het programma gemaakt is wordt de cryptische maar tevens realistische foutmelding “parse error” in het uitvoerwindow afgedrukt. Om de nieuwe gebruikers niet al te veel met computerterminologie te overdonderen wordt de plaats in het programma waarin de fout gemaakt is niet door middel van een regelnummer aangegeven, maar doordat de foutmelding ook daadwerkelijk op de plaats waar de turtle zich toen bevond wordt afgedrukt.

We geven de gebruiker de opdracht om te proberen een bepaalde gewenste output te genereren. Terwijl de opdracht wordt uitgevoerd meten we het aantal compilatiecycli dat doorlopen moet worden voordat een bevredigend resultaat wordt bereikt (dit wordt beoordeeld door één van de onderzoekers). Ook kijken we naar de tijdsduur van de afzonderlijke ontwerpcycli. Hieruit kunnen we wellicht bepaalde leerpatronen afleiden, bijvoorbeeld of een gebruiker veel korte cycli prefereert boven een kleiner aantal langere, meer doordachte cycli.

Graphical Programmeertest

In deze test wordt het grafische interface volledig gebruikt. Zichtbaar zijn nu nog steeds de commandotekst- en uitvoer-windows, maar nu wordt de prominente plaats ingenomen door een grafisch-geöriënteerd commando window (“Commands”) dat de functie van de TRIP Structure Manager simuleert. Bovendien verschijnt er een nieuw gelijknamig menu in de menubalk waarin alle door de simulatortaal ondersteunde commando's beschikbaar zijn. We laten de commandotekst window opzettelijk staan om te benadrukken dat men nu niet ineens met een geheel ander systeem werkt, maar dat het grafische interface het eigenlijke programma voor de gebruiker genereert. Zo wordt het de gebruiker duidelijk wat precies het nieuwe systeem voor hem doet.

De bedoeling is nu wederom dat de gebruiker een opdracht uitvoert, maar dat in plaats van het rechtstreeks intypen van regels tekst de commando's in de vorm van iconen gemanipuleerd kunnen worden. Elk icoon stelt één bepaald commando voor. Nieuwe commandoiconen kunnen aangemaakt worden door een overeenkomstige selectie uit het menu te maken. Aangezien de commando's in een wel-gedefinieerde volgorde doorlopen moeten worden geven we deze met behulp van verbindende lijnen aan.

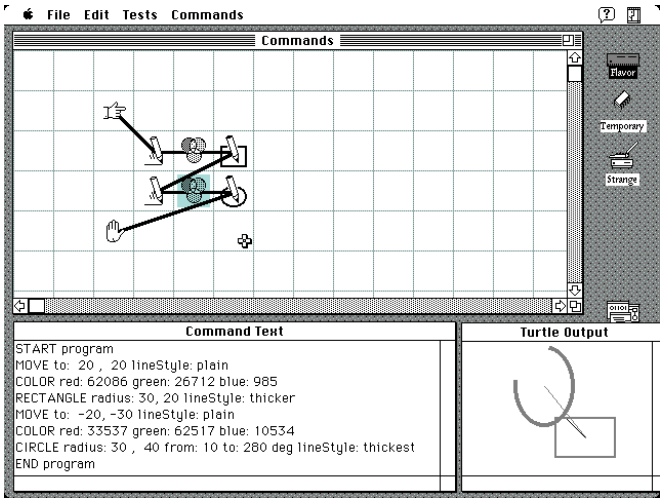


Figuur 19. Voorbeeld van een dialoog

Iconen kunnen geselecteerd worden door erin te klikken. Voor ervaren gebruikers bestaan er keyboard shortcuts— overeenkomstig het door Apple vastgelegde gedrag dat een gebruikers-interface moet vertonen heeft een druk op de “Tab” toets tot gevolg dat het volgende icoon geselecteerd wordt, en “Shift-Tab” (een druk op “Tab” terwijl de “Shift” toets ingedrukt is) het vorige icoon. Alhoewel er gebruikersinterface-bepalingen bestaan voor het maken van “multiple selections”, d.i., waarbij meerdere entiteiten gelijktijdig geselecteerd kunnen worden, achtten wij deze uitbreiding en de consequenties voor een eerste kennismaking te complex en bovendien voor de evaluatie weinig relevant.

Iconen kunnen door de gebruiker willekeurig (maar vastgepind aan een grid) over het veld verplaatst worden. Dit heeft geen consequenties wat betreft de uitvoer.

Een dubbel-klik op een icoon (of een druk op de “Return” of “Enter” toets) heeft tot gevolg dat de parameterdialoog bij dat commando geopend wordt. Hierin kan de gebruiker naar believen de commandoparameters bezichtigen en modificeren met behulp van “edit text items” en pop-up menus. Deze zorgen voor impliciete correctheid van de gegenereerde commando's, omdat het gewoonweg niet mogelijk is om in zo'n dialoog voor



Figuur 18. Voorbeeld van de Graphical Programmeertest

het commando onjuiste parameters te specificeren. (Het is echter nog wel mogelijk om in het geval van een numerieke parameter een waarde te specificeren die buiten het toegestane bereik van de parameter ligt. Dit in tegenstelling tot de echte TRIP, waarin mechanismen zijn ingebouwd die dit voorkomen.)

Om redenen van tijd en om wederom het geheel niet al te complex te maken hebben we gekozen voor een modale dialoogstructuur. Dat wil zeggen dat er maar één commandodialoog tegelijk geopend kan zijn, en dat deze eerst afgehandeld moet worden voordat doorgegaan kan worden met de rest van het programma. Onder andere heeft dit tot consequentie dat de “Confirm” en “Revert” buttons zoals in ons werk beschreven niet nodig zijn. In plaats daarvan volstaan nu volledig de gebruikelijke twee buttons die het werken met een dialoog beëindigen: een klik op “OK” (of een druk op “Return” of “Enter”) accepteert de eventueel in de dialoog gemaakte veranderingen en voert deze door in het programma (zoals ook in de textuele representatie van het programma te zien is), terwijl “Cancel” (of een druk op “Command-.” † of “Esc”‡) de dialoog sluit zonder dat gemaakte veranderingen in het programma ook daadwerkelijk doorgevoerd worden.

Nieuwe commando's worden toegevoegd vóór het huidig geselecteerde icoon. Wanneer er geen icoon geselecteerd is dan wordt het nieuwe commando aan het eind van het programma toegevoegd. Dit lijkt de juiste keuze omdat op deze manier, als de gebruiker geen enkele selecties maakt, het programma in precies die volgorde opgebouwd wordt waarin de gebruiker de menuselecties maakt.

Door het zichtbaar blijven van de commandotekst window wordt de nauwe binding met een commandoicoon en zijn bijbehorende commandoregel benadrukt. Bovendien geeft het voor de gebruiker duidelijker aan wat de eigenlijke rol van het grafische interface is, d.i. niet zozeer iets compleet anders wat nieuwe stof tot leren geeft, maar een hulpmiddel dat als vereenvoudiging van de programmainvoer dient.

We verwachten dat uit deze test zal blijken dat het aantal syntaxfouten en de daarmee gerelateerde compilatiecycli significant zal afnemen.

---

† “Command” is op Apple keyboards de ◢ toets

‡ Hierbij wordt benadrukt dat “escape” inderdaad zoals het woord ook suggereert een ‘veilige uitgang’ behoort te bieden

Onderzoek

In dit hoofdstuk wordt de uitvoering van het onderzoek beschreven. Hierbij wordt nadrukkelijk aandacht besteed aan de voor de evaluatie van het gebruikersinterface te gebruiken methodologie. Zoals in het vorige hoofstuk al beschreven is wordt hierbij gebruik gemaakt van vier verschilende tests die ieder verschillende aspecten testen.

METHODOLOGIE

[Apple88c] geeft een tien-staps methodologie voor het houden van gebruikersonderzoek. We hebben deze gevolgd voor het eigen onderzoek. [Apple85, pp. V-25] geeft verdere informatie over het user-testing proces.

1. *Introduce yourself*
2. *Describe the purpose of the observation in general terms*

   Het is belangrijk duidelijk te maken dat niet de gebruiker maar het programma getest wordt— er wordt juist gezocht naar moeilijk te gebruiken aspecten van het programma, het is geen IQ-test. Met andere woorden, als de gebruiker moeite heeft met het uitvoeren van de opdracht ligt dat niet aan hemzelf maar aan het programma.
3. *Talk about the equipment in the room*
4. *Explain how to 'think aloud'*

   Omdat het primaire doel van de gebruikerstest het evalueren van TRIP was, wilden we zoveel mogelijk nuttige informatie verzamelen over het gebruik ervan. Een uitstekende manier om dit te doen is de gebruiker hardop te laten denken. Nuttige evaluatieresultaten hoeven immers niet noodzakelijkerwijs quantificeerbaar te zijn. Hierdoor kunnen misconcepties of vooroordelen die de schrijvers van het programma hebben aan het licht komen. Het is zeer goed

mogelijk dat een gebruiker die het programma voor het eerst ziet het op een totaal verschillende manier beschouwt.

Omdat het 'hardop denken' door veel mensen als 'raar' wordt ondervonden is het misschien nodig ze op dit punt gerust te stellen of het eventueel voor te doen. Als de gebruikers vergeten hardop te denken tijdens de opdracht moeten ze hieraan herinnerd worden.

5. *Explain that you will not answer questions or provide help*

   Zoals gezegd is het belangrijk dat de deelnemers tijdens het uitvoeren van de opdracht zonder storing of hulp hun opdrachten uitvoeren. Alleen op deze manier kan nuttige informatie over het gebruik van het programma worden verworven. Hoewel de begeleider geen vragen van de gebruiker kan beantwoorden, benadrukken we dat we echter wel willen dat ze ze stellen (in verband met het vorige punt).

   Wel stellen we van tevoren dat als een gebruiker na een aantal minuten niet verder komt, de opdracht afgebroken en als niet succesvol afgerond beschouwd wordt.

6. *Describe the tasks*

   We leggen uit wat verwacht wordt en geven een geschreven vorm van de opdracht. Verder stellen we extra documentatie ter beschikking die tijdens het uitvoeren gebruikt kan worden. Zoals eerder gezegd willen we realistische situaties testen, niet het geheugen van de deelnemer. In werkelijke situaties staat immers ook altijd de documentatie ter beschikking.

7. *Show the participant how to use any hardware that may be unfamiliar, and introduce concepts that are prerequisites to the task*

   Alhoewel alle deelnemers worden verondersteld voldoende bekend te zijn met het gebruik van de Macintosh Computers en in het bijzonder van een WIMP interface verzekeren we ons hiervan door ze de gelegenheid te geven vragen hierover te stellen:

8. *Ask if there are any questions before you start; then, begin the observation*

9. *Conclude the observation*

   Merk op dat er geen interventie is tijdens het uitvoeren van de opdracht. Als deze tot een einde is gekomen leggen we de gebruiker uit wat we testten, beantwoorden we eventuele vragen die tijdens de opdracht zijn gesteld en geven we nogmaals de gelegenheid om opmerkingen te maken over de opdracht en de gebruikte software.

10. *Use the results*

    De testresultaten worden in de hierna volgende delen geëvalueerd. Hierbij is het belangrijk te realiseren dat eventuele plaatsen waar de gebruiker moeite had niet geweten kunnen worden aan het gebrek aan ervaring met het systeem, maar aan het systeem zelf.

EVALUATIE

Het doel van de evaluatie is om te zien of en in hoeverre het TRIP systeem gebruikers van simulatieprogramma's helpt bij het schrijven van hun simulatiespecificaties. We hebben hierbij een aantal criteria die van belang zijn [Foley90 p.391]:

- *Leercurve* Dat wil zeggen, de tijd die het kost om het systeem te leren;
- *Snelheid* Hoe lang duurt het daarna om een specificatie te schrijven?
- *Nauwkeurigheid* Hoe correct is dan de gegenereerde specificatie?
- *Werkstijlen* Hoe werkt de gebruiker met het systeem?
- *Aantrekkelijkheid* Het blijkt dat in onderlinge vergelijkingen gebruikers soms liever met een systeem werken dat niet persé het snelste of makkelijkste is.

Deze criteria volgen uit de probleemidentificatie die aan het begin van dit verslag is gegeven. Een criterium dat niet in de evaluatie is opgenomen is:

- *Recall* Hoe snel wordt men weer vertrouwd met het systeem na het een langere tijd niet gebruikt te hebben?

De instructies voor de uit te voeren taken moeten uiteraard duidelijk zijn, maar niet zover gaan als uit te leggen hoe de opdracht uitgevoerd kan worden, anders zouden de testresultaten weinig zinvol zijn.

REPRESENTATIEVE TESTSUBJECTEN

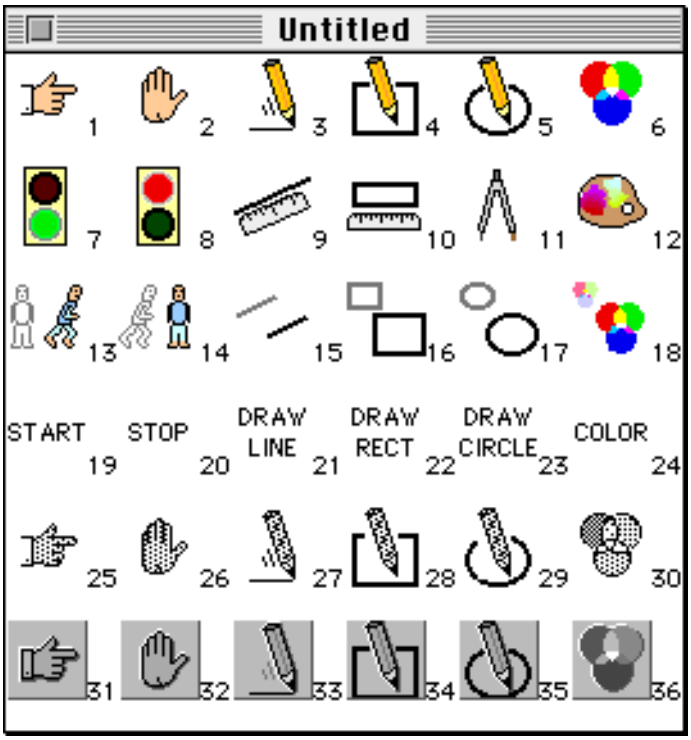
De keuze van de testgebruikers is van groot belang. Ten eerste is het om statistisch significante uitspraken te kunnen doen noodzakelijk om over een voldoende grote groep proefpersonen te kunnen beschikken. Met name dit is voor ons een probleem. Hierbij moet er verder erop gelet worden dat de testgebruikers tot de doelgroep behoren. In ons geval is deze groep zeer divers zodat het aantal testgebruikers niet verder beperkt wordt.

We maken voor sommige uitspraken in het onderzoek onderscheid tussen twee soorten gebruikers: ‘ervaren’ en ‘onervaren’. Ervaren gebruikers beschouwen we als zijnde diegenen die gebruik maken van programma's waarbij sprake is van verwerking door de computer van textuele invoer. Hierbij is bijvoorbeeld wel inbegrepen iemand die C programma's schrijft, maar niet iemand die een computer gebruikt voor tekst‘bewerking’, omdat hier niet van *ver*werking van gegevens door de computer sprake is. Het schrijven van programma's toont sterke overeenkomsten met het schrijven van simulatorspecificaties in de regelmatige *write-compile-debug* cycli. Een alternatief zou geweest zijn om in plaats van een indeling in discrete groepen gebruik te maken van een continue verdeling. Hiervoor zouden we de beschikking moeten hebben over een betere methode om ervarenheid te quantificeren. De icoonherkenningstest en de icoonselectietest zijn uitgevoerd door alle testgebruikers. Voor de uitvoering van de twee programmeertests hebben we slechts gebruik kunnen maken van een iets beperktere groep. Voor deze laatste twee tests hebben we tevens de groep testgebruikers in twee min of meer gelijke groepen verdeeld. Hierbij is gebruik gemaakt van informele criteria om de ervarenheid van gebruikers in te schatten.

We hebben de eerste twee opdrachten door 16 gebruikers laten uitvoeren. Hiervan waren er 6 ervaren en 10 onervaren gebruikers. De onervaren gebruikers variëerden van matig ervaren (techniekstudenten) tot onervaren computergebruikers. In overeenstemming met punt 7 uit de bovenstaande methodologie zorgden we er wel voor dat alle gebruikers alvorens met de opdrachten te beginnen voldoende bekend zijn met het gebruikte computersysteem, in dit geval de Apple Macintosh. Dit leverde overigens in het algemeen geen probleem op. Voor de programmeertests hebben we gebruik gemaakt van 10 testgebruikers die verdeeld werden over twee min of meer gelijke groepen, zodat we de resultaten van deze beide groepen met elkaar kunnen vergelijken.

DE ICOON HERKENNINGSTEST

De testgebruikers wordt verteld dat we onderzoek doen naar het gebruikersinterface voor een tekenprogramma. Dit is geoorloofd aangezien een gebruiker normaal gesproken ook enig idee heeft wat voor soort programma hij gaat gebruiken. We vertellen dat zij een



Figuur 20. De iconen die gebruikt zijn voor de evaluatie

veld met 36 verschillende genummerde plaatjes (*iconen*) te zien krijgen. Het is de bedoeling op te schrijven wat de gebruiker denkt dat de iconen in het programma zouden bewerkstelligen, *niet* wat de iconen zelf voorstellen. Hierbij vermelden we tevens dat het mogelijk is dat meerdere iconen het zelfde voorstellen. Hiermee hopen we te voorkomen dat de gebruikers zich al te veel gaat concentreren op het zoeken van verschillen tussen gelijkende iconen.

We verzoeken de gebruikers eventuele correcties niet weg te gummen maar leesbaar door te strepen zodat het later beter mogelijk is om hun gedachtengang te reconstrueren. Bovendien hoeven de antwoorden niet in goed Nederlands/Engels te zijn, omdat het gaat om de *stream-of-consciousness* en niet de grammatica. Als een icoon niet te begrijpen valt is dat evengoed een 'juist' antwoord— het doel van het onderzoek is immers om er achter te komen welke iconen inuïtief te begrijpen zijn. Hierbij onderstrepen we tevens dat we dat we het programma onderzoeken en niet de gebruiker.

Na afloop worden de antwoorden en de iconen met de gebruiker besproken zodat deze de tijd krijgt om de betekenis te leren kennen. Dit is van belang voor de volgende test omdat we daar juist het herkennen van iconen willen testen. Tevens wordt de gebruiker in staat gesteld om eventuele andere opmerkingen over de opdracht te maken.

Subjectieve evaluatie

Figuur 20 toont de gebruikte iconen. Nogmaals, in de eigenlijke test was de volgorde van de iconen willekeurig.

Directe methode. De eerste groep van iconen (in de figuur genummerd 1-6) waren ontworpen volgens de directe methode. Deze methode leverde relatief weinig herkenningsproblemen op.

Het lijntekenicoon werd soms opgevat als het maken van een arcering in zwart-wit. Het icoon zou verbeterd kunnen worden door de bewegingsstreepjes te verwijderen. Ook werd het stopteken soms opgevat als omhoog gaan. In de rechthoek- en cirkel-teken

iconen werd soms het potlood opgevat als het aangeven van een specifiek punt op de figuur, of als het van binnenuit inkleuren. Al deze problemen lijken opgelost te kunnen worden door overbodige elementen (het potlood, de bewegingsstreepjes) uit de iconen te verwijderen. Dit is overigens in overeenstemming met de aanbevelingen van Apple ten aanzien van het ontwerpen van iconen.

Het enige icoon dat echt een probleem opleverde was het kleurselectie icoon. Kennelijk is deze voorstelling van de rood, groen, blauw en mengkleuren niet echt duidelijk voor met name beginnende gebruikers.

Indirecte methode. De voor de tweede set [7-12] gebruikte methode wordt ook wel *object-functie* genoemd. Gebruikers hadden duidelijk problemen met het interpreteren van deze iconen.

Zo werden bijvoorbeeld de iconen die stoplichten voorstellen en dus de start en stop commando's representeren soms opgevat als kleurselecties. Iemand vatte het stoplicht op als “wachten” in plaats van “stoppen” en het groene licht als “doorgaan” in plaats van “beginnen”.

Het kleurselectie icoon werd soms geïnterpreteerd als het selecteren van een (klein) gebiedje. Één van de gebruikers stelde echter dat als de vormgeving van het schilderspalet beter (natuurgetrouwer) was geweest deze wel herkend zou zijn.

De iconen voor het tekenen van lijnen en rechthoeken waren misschien wat ongelukkig gekozen. Het probleem is dat er niet echt een voorwerp is dat specifiek gebruikt wordt voor het tekenen van een rechthoek. Eventueel was de keuze van een geo-driehoek in plaats van een normale lineaal beter geweest. De lineaal werd door één gebruiker in eerste instantie aangezien voor een kam. De gebruiker concludeerde echter uit de context van de opdracht na enig denken toch dat het een lineaal was die in de lijn-teken icoon gebruikt werd om de diagonaal te meten en in het rechthoek-teken icoon om de breedte te meten. De gebruiker kiest hier duidelijk voor een te directe interpretatie. Interessant is echter dat het passericoon daarentegen geen enkel probleem opleverde.

Voor/na-methode. De iconen uit de derde groep (13-18) pogen de situatie vóór en na het uitvoeren van het commando weer te geven.

De start- en stop-iconen werden vrij goed geïnterpreteerd, zij het soms iets te direct door letterlijk een beweging te veronderstellen. Vrijwel alle gebruikers herkenden dat de iconen uit twee delen bestonden en onderkenden tevens dat de delen van de icoon met een hogere kleurverzadiging de belangrijkste waren.

De cirkel- en rechthoek-iconen werden zo goed als unaniem gemisinterpreteerd. Ofwel werden ze geassociëerd met een verandering in grootte van het getoonde object of met het copiëren van een object. De verdubbeling van het object binnenin een enkel icoon komt niet overeen met dat het slechts om één object gaat. Deze resultaten duiden op een verkeerde toepassing van de ontwerpmethode, maar dat deze methode indien juist gebruikt wel effectief kan zijn.

Het lijn-icoon en het kleurselectie-icoon werden soms geïnterpreteerd als een verandering in kleurintensiteit en in eengeval als het aanbrengen van een schaduw. Hier blijkt weer hoe nuttig het uitvoeren van een gebruikerstest is. Vaak werden we zeer verrast door de interpretatie van de iconen door de gebruiker.

Tekst-directe methode. Enigzins tot onze verbazing bleken zich ook hier (19-24) een aantal problemen voor te doen. Zo bleek soms het Engels en de gebruikte afkorting een probleem op te leveren. Waarmee weer eens is aangetoond dat niets vanzelfsprekend is en een foutieve veronderstelling snel gemaakt is.

Gesimuleerde-kleur directe methode. Bij deze methode zijn de kleuren van de iconen (25-30) van de directe methode vervangen door diverse arceringen. We toetsten hiermee de uitspraak dat het simuleren in zwart-wit van kleuren of grijstinten verstorend zou zijn.

De gesimuleerde-kleur directe iconen leverden eigenlijk vrijwel dezelfde resultaten op als de normale directe, met de uitzondering dat het kleurenselectie-icoon nu soms als

het selecteren van een arcering geïnterpreteerd wordt. Dit is waarschijnlijk moeilijk te voorkomen.

We kunnen hieruit concluderen dat het gebruik van kleur voor de herkenning van iconen beslist niet altijd noodzakelijk is. Of het gebruik van kleur ook tot een meetbare verbetering van het gebruikersinterface leidt (op anders dan zuiver esthetische gronden) is beslist niet zeker. In de icoon selectie test wordt hier verder onderzoek naar gedaan.

3D-effect methode. Het toevoegen van een drie-dimensionaal effect aan de iconen (31-36) levert geen extra problemen op in vergelijking met de normale directe iconen. Met één uitzondering waar de testgebruiker het 3-dimensionale effect interpreteerde als het tekenen binnen een kader. Geen van de test-gebruikers sprak voorkeur uit voor de 2- of de 3-dimensionale versies. Dit is op zich interessant, omdat een aantal nieuwe interface toolkits (waaronder MS-Windows en OSF/Motif) uitgebreid gebruik maken van 3D-effecten. Terwijl in tijdschriften en uit gesprekken met users vaak twijfel klinkt over het nut ervan. Vanuit het oogpunt van de programmeur of ontwerper van het interface zitten er een aantal nadelen aan vast, zoals grotere complexiteit van de graphics en een vergroot gebruik van schermoppervlak. In het dagelijks leven wordt juist vaak gekozen voor een meer abstracte representatie in plaats van te vervallen in veel detail, denk bijvoorbeeld aan verkeersborden. Ook onze resultaten met de directe iconen wijzen op de noodzaak voor een zo simpel mogelijk icoonontwerp.

Samenvatting. Aan de hand van de gevonden resultaten kunnen we opnieuw een aantal richtlijnen voor het ontwerpen van iconen opstellen. In de eerste plaats, houdt het simpel: voorkom het gebruik van onnodige elementen in een ontwerp. Wees voldoende nauwkeurig in het afbeelden van objecten en zorg dat de juiste en voldoende details getoond worden maar verwijder overbodige details. In het geval dat van de gebruiker verwacht wordt een vergelijking te maken tussen verschillende elementen zorg er dan voor dat alleen de bedoelde parameter veranderd is. Het gebruik van kleur draagt niet noodzakelijk bij aan de herkenbaarheid van iconen. Een mogelijke uitzondering daarbij vormt de selectie van een kleur. Ook 3D-effecten lijken niet aan de herkenbaarheid bij te dragen.

Achteraf gezien was het waarschijnlijk verstandiger geweest om de iconen sequentiëel aan de gebruiker te tonen in plaats van in een groot veld. Het was nu niet helemaal te voorkomen dat de gebuiker verschilende iconen met elkaar ging vergelijken. Bijvoorbeeld werd soms een 3D-variant geïnterpreteerd als een ‘tegenstelling’ of als het ‘opheffen’ van het corresponderende directe icoon.

Concluderend kunnen we vaststellen dat onze resultaten de in de literatuur beschreven conclusies bevestigen. Deze test heeft bovendien onze ogen geopend voor de moelijkheid van het voorspellen van het gedrag van testgebruikers. In de toekomst zullen we zeker van deze kennis gebruik kunnen maken bij het ontwikkelen van nieuwe software.

DE ICOON SELECTIETEST

Deze test dient net als de vorige twee doelen. In de eerste plaats helpt ze ons beter inzicht te krijgen in het gebruik van een iconische representatie en in de tweede plaats vormt ze een introductie tot de programmeertests.

De gebruikers wordt verteld dat het bij deze test de bedoeling is om bij de gepresenteerde commandobeschrijving het bijbehorende icoon te selecteren. De gebruiker wordt steeds één van de zes commando's gepresenteerd waarna hij gevraagd wordt om het bijbehorende icoon uit het volledige palet van zes volgens een bepaalde methode ontworpen iconen te selecteren. We behandelen de volledige verzameling van $6 \cdot 6$ iconen. De volgorde waarin de commando's gepresenteerd worden en de plaats van de iconen in het palet is iedere keer willekeurig. De handeling die nodig is om de iconen te selecteren is een simple muis beweeg- en klik-operatie. We benadrukken dat als de gebruiker vragen heeft deze alleen nu beantwoord kunnen worden. Tijdens deze

test is het van minder belang dat de gebruikers hardop denken om hun keuzes te rechtvaardigen omdat we in de vorige test reeds voldoende informatie over het herkennen van iconen hebben verzameld. Zoals eerder vermeld wordt de gebruikers niet verteld dat we reactietijdmetingen doen.

Objectieve evaluatie

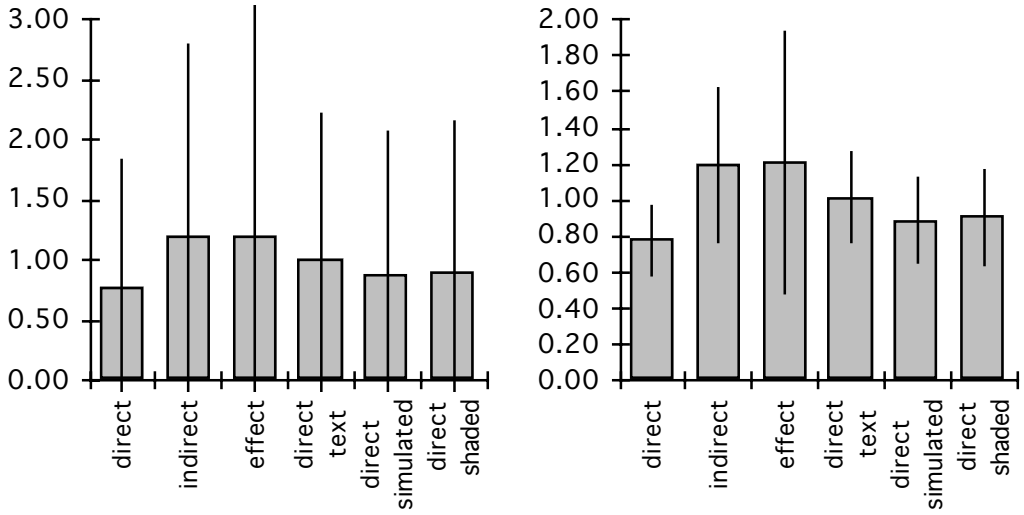
Het programma meet de tijd die een gebruiker nodig heeft om de juiste icoon bij het gepresenteerde commando te selecteren. Deze meetgegevens zijn gebruikt om te analyseren welke methode van iconische representatie leidt tot de beste resultaten. In eerste instantie dachten we naast snelheid ook het aantal gemaakte foutieve selecties te kunnen onderzoeken. Het bleek echter dat gebruikers op een enkele uitzondering na geen foutieve selecties maakten. Dit duid er weer op dat de herkenning van de iconen niet veel problemen opleverden. In de volgende sectie bespreken we de methode, en de statistische onderbouwing ervan, die we hebben gebruikt voor de analyse van de meetgegevens.

De belangrijkste vraag waar we een antwoord op zoeken is welke manier van iconische presentatie zorgt voor de snelste uitvoering van de opdrachten, dat wil zeggen de snelste selectie van een commando. We hebben reeds geconstateerd dat fouten nagenoeg niet voorkomen.

We willen weten welke methode van representatie voor de grootste groep gebruikers relatief het beste werkt. Waar we nadrukkelijk niet in geïnteresseerd zijn is de absolute reactiesnelheden omdat deze per gebruiker naargelang muiservaring kunnen verschillen. Deze verschillen kunnen we 'uitfiltreren' door de resultaten per individuele gebruiker te normaliseren. Normaliseerden we niet, dan zou dit bijvoorbeeld inhouden dat de resultaten van één langzame gebuiker veel zwaarder zouden tellen. De normalisatie maakt dat de gemiddelde selectieduur voor alle iconen per gebruiker gelijk is aan 1. Om tot een uiteindelijke uitspraak te kunnen komen over welke representatiemethode het beste is wordt het gemiddelde van de diverse genormaliseerde selectieduren per methode over alle gebruikers genomen. Tevens worden tegelijkertijd de varianties in de metingen doorberekend.

Figuur 21 toont deze gemiddelden van de genormaliseerde resultaten. In alle grafieken zijn we uitgegaan van onzerkerheidsintervallen voor een betrouwbaarheid van 80%. We hebben uit interesse in deze figuur de onzekerheidsintervallen berekend volgens de ongelijkheid van Tschebyshev en volgens een normale $N(\mu, \sigma^2)$ verdeling, [Smit87]. Tschebyshev geldt voor een willekeurige distributie en levert daarom in het algemeen een worst-case afschatting van de fout. Volgens de Centrale Limietstelling mag echter voor een voldoende grote populatiegrootte een normale verdeling verondersteld worden, en met behulp een tabel voor een standaard normale verdeling kan dan een afschatting voor de afwijking worden gevonden. Bij het berekenen van de varianties in de stochasten betrekking hebbend op de totale populatie [Sloff77] zijn we ervan uitgegaan dat de resultaten van de afzonderlijke gebruikers statistisch paargewijs onafhankelijk waren.

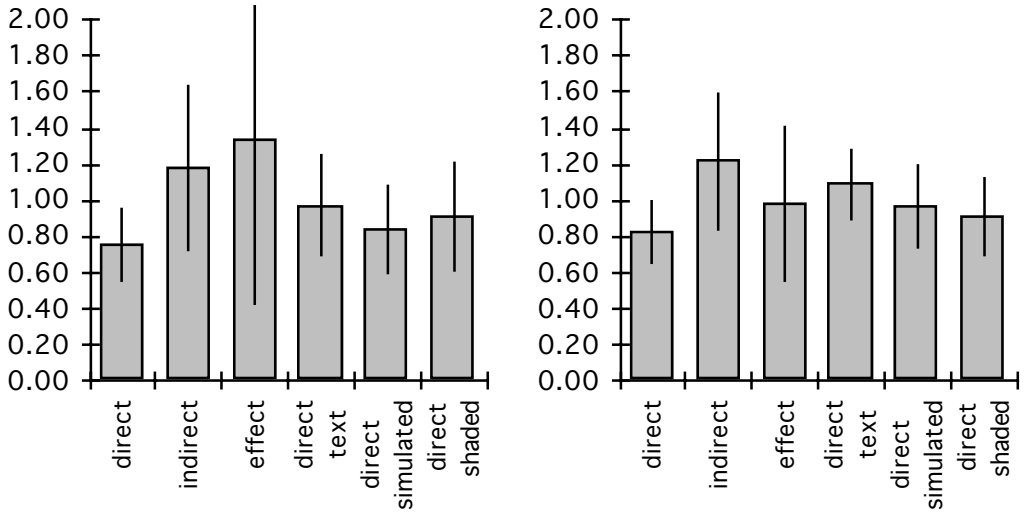
Voor de verwerking van de resultaten is gebruik gemaakt van een aantal spreadsheets. Deze zijn geschikt voor Excel voor de Apple Macintosh en zijn op verzoek beschikbaar bij de auteurs.



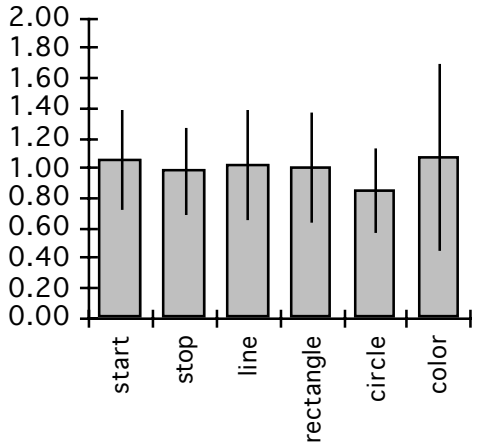
Figuur 21. Gemiddelde genormaliseerde selectiesnelheid naar icoontype, met onzekerheidsintervallen voor (links) een Tschebyshev afschatting en (rechts) een normale verdeling, 80% betrouwbaarheid

Uit de normale verdeling valt af te leiden dat, met 80% betrouwbaarheid, directe iconische representatie een 0.8-2.1 keer zo hoge reactiesnelheid oplevert als de directe tekstuele.

Interessant is nu ook de vraag of voor zowel ervaren als onervaren gebruikers de diverse ontwerpmethoden dezelfde resultaten opleveren. Nadeel van het opsplitsen in twee groepen is natuurlijk dat de groepen erg klein worden en we nog voorzichtiger moeten zijn met het trekken van conclusies. De resultaten van deze splitsing wordt getoond in figuur 22.



Figuur 22. Gemiddelde genormaliseerde selectiesnelheid naar icoontype, voor (links) onervaren en (rechts) ervaren gebruikers, onzekerheidsintervallen voor een normale verdeling, 80% betrouwbaarheid



Figuur 23. Gemiddelde genormaliseerde selectiesnelheid naar type commando, onzekerheidsintervallen voor een normale verdeling, 80% betrouwbaarheid

Het eerste dat opvalt is dat de onderlinge verschillen in reactiesnelheid voor de ervaren gebruikers kleiner zijn. Bovendien zijn de onzekerheidsintervallen voor de ervaren gebruikers kleiner zijn dan die voor de onervaren gebruikers. Het lijkt er dus op dat de selectiesnelheid van gebruikers al naargelang de ervaring convergeren. Het is voor de ervaren gebruikers mogelijk om statistisch verantwoorde uitspraken te doen over de verschillen tussen de diverse ontwerpmethoden. Zo is de directe ontwerpmethode significant sneller dan de methode die gebruik maakt van tekst.

Tot slot hebben we nog gekeken of er bepaalde commando's zijn die meer problemen opleveren. Zo zijn bijvoorbeeld de start en stop commando's duidelijk abstracter dan de teken rechthoek etc. commando's. De resultaten van deze evaluatie is getoond in figuur 23.

Er zijn geen significante verschillen waar te nemen tussen de diverse commando's. Het hoge gemiddelde en het grote onzekerheidsinterval bij het kleurselectie commando doet vermoeden dat hier iets aan de hand is zoals een probleem icoon of een toevallige extreme meting. Bij nadere beschouwing van de meetresultaten blijkt inderdaad dat twee gebruikers een ‘uitschieter’ hebben bij dit commando.

Ook hier blijken de resultaten van de ervaren gebruikers consistenter te zijn en lijkt convergentie op te treden. Ze lijken ons vermoeden dat iconen voor abstractere begrippen moeilijker te herkennen zijn in ieder geval niet tegen te spreken. Misschien komt dit doordat ze moeilijker door iconen weer te geven zijn.

Het trekken van conclusies is bij deze test vrij moeilijk omdat er erg veel testgebruikers nodig zijn om statistisch significante uitspraken te kunnen doen. Dit is voor ons een probleem omdat wij daar niet over kunnen beschikken. De resultaten duiden echter wel duidelijk in een richting. Het gebruik van directe iconen in sneller dan het gebruik van tekst en bovendien lijkt ook het gebruik van kleur de selectie snelheid te vergroten.

DE PROGRAMMEERTESTS

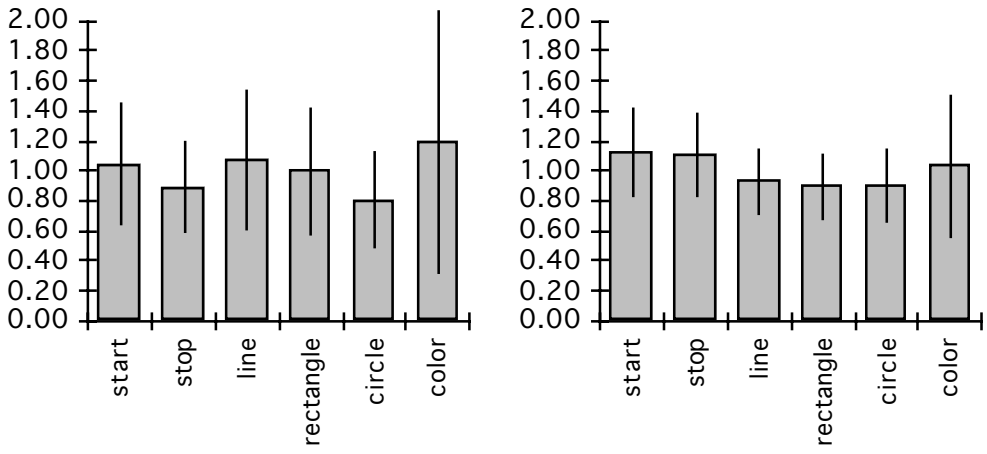
Om TRIP aan de hand van de aan het begin van dit hoofdstuk genoemde zes criteria te toetsen hebben we een aantal taken ontworpen die we aan testgebruikers hebben voorgelegd. We proberen hierbij zoveel mogelijk *realistische* opdrachten te geven zodat het *hele* proces getest wordt en niet één klein aspect dat misschien belangrijk lijkt. Om de resultaten te kunnen relateren aan de situatie waarbij de gebruiker niet de beschikking heeft over een grafisch interface laten we ook een serie vergelijkbare opdrachten uitvoeren waarbij geen grafisch interface gebruikt wordt maar een eenvoudige tekstueel interface (zij het ook een direct-manupulatie interface en niet zoiets primitiefs als de UNIX utility '`vi`').

De opdrachten en daarbij behorende gebruikersinstructies staan in appendix C. De lezer wordt verzocht deze appendix nu eerst te lezen zodat zij bekend is met de opdrachten. Het is uiteraard belangrijk voor het uitvoeren van de taken dat de instructies duidelijk zijn, maar niet zover gaan als uit te leggen hoe de opdracht uitgevoerd kan worden, anders zouden de testresultaten weinig zinvol zijn.

De programmeeropdracht bestaat uit twee equivalente series van vijf opdrachten, waarbij de ene serie met behulp van het grafische en de andere met behulp van een tekstinterface wordt ingevoerd. De groep testgebruikers wordt daartoe gesplitst in twee vergelijkbare groepen— hiermee bedoelen we dat beide groepen zijn opgebouwd uit gebruikers die paarsgewijs een vergelijkbaar ervaringsniveau bezitten. Dit maakt het mogelijk om de resultaten van beide groepen te vergelijken. Hier moet echter de nodige voorzichtigheid worden betracht aangezien de gebruikers natuurlijk nooit precies gelijk zijn. Dit heeft een storend effect op de meetresultaten, zeker wanneer slechts gebruik gemaakt kan worden van een zeer beperkte groep testgebruikers.

Bovendien hebben we het aantal opdrachten moeten beperken tot 5. Liever zouden we natuurlijk de testgebruikers meer opdrachten hebben laten uitvoeren zodat we nauwkeuriger resultaten voor bijvoorbeeld de leersnelheid kunnen verkrijgen. Dit is echter niet mogelijk omdat het dan helemaal erg moelijk is om testgebruikers te vinden die bereid zijn de benodigde tijd te investeren.

De opdrachten in de twee series zijn geordend oplopend in moeilijkheidsgraad. De eerste opdracht is erg simpel en is hoofdzakelijk bedoeld om de gebruiker vertrouwd te maken met het systeem zodat in de vervolgopdrachten minder storende effecten zullen



Figuur 24. Gemiddelde genormaliseerde selectie snelheid naar type commando voor (links) onervaren en (rechts) ervaren gebruikers en onzekerheidsintervallen voor een normale verdeling en een betrouwbaarheid van 80%

optreden door een (toevallig) verschil in ingangsniveau. Elke serie bevat bovendien twee vrijwel identieke opdrachten (de rechthoeken) waarmee we het veranderende ervaringsniveau van de gebruiker willen meten. We veronderstellen dat de gebruiker meer ervaren wordt naarmate hij meer opdrachten heeft uitgevoerd. Dit (relatieve) ervaringsniveau willen we bepalen door de snelheid waarmee een gebruiker deze twee *referentieopdrachten* uitvoert te meten. Het ervaringsniveau van de gebruiker modeleren we als een eenvoudige exponentiële functie en de daarmee gerelateerde uitvoeringssnelheid als een afnemende exponentiële functie die voor een ervaringsniveau van $+\infty$ naar een constante limiet convergeert. Met behulp van de referentieopdrachten kunnen we nu deze limiet voor zowel de grafische als voor de tekstuele programmeermethode onderzoeken en op basis daarvan een uitspraak doen over de efficiency van beide methoden voor ervaren gebruikers. Tevens is de tijdconstante van deze exponentiële functie een indicatie voor de snelheid waarmee dit eindniveau wordt bereikt.

De overige drie opdrachten hebben een oplopende moeilijkheidsgraad. Hierdoor hopen we, naast een hoger ervaringsniveau, te bereiken dat de gebruiker fouten gaat maken en daardoor edit-acties moet uitvoeren zodat de volledige ontwerp-debug cyclus doorlopen moet worden. De resultaten van de grafische tests kunnen we weer vergelijken met die van de tekstuele tests. Hierdoor hopen we uitspraken te kunnen doen over welke methode van specificatie de beste is. Het begrip 'beste' omvat hierbij ondermeer *snelheid*, *nauwkeurigheid*. Daarnaast worden ook de criteria: *werkstijl* en *aantrekkelijkheid* in de evaluatie meegenomen.

Turtle taal

Tot slot nog iets over de Turtle taal. We hebben besloten een nieuwe taal voor de evaluatie van TRIP te ontwerpen omdat de invoertaal die door de simulator TRENDY gebruikt wordt alleen begrijpelijk is voor electrotechnici die gespecialiseerd zijn in proces- en devicesimulatie. Indien we geen nieuwe taal gebruikt hadden zou onze groep potentiële testgebruikers onnodig beperkt worden. Om deze reden hebben we een nieuwe taal ontworpen, de Turtle taal. Deze is zo ontworpen dat hij de syntactische en grammaticale moeilijkheden zoals die in simulatietalen voorkomen bevat. Alleen is nu gekozen voor een abstractie die voor een breder publiek begrijpelijk is.

In de Turtle taal is onder meer een grote gevoeligheid voor spaties, hoofdletters en returns ingebouwd. Dit moge bijvoorbeeld blijken uit de verplichte return na het laatste "END program" commando. Bovendien zijn er extra problemen ingebouwd, zoals inconsistentie tussen commando's. Denk hierbij bijvoorbeeld aan de absolute coordinaten waar in het "MOVE" commando gebruik van worden gemaakt in vergelijking met de relatieve (straal) coordinaten in het cirkel commando. We hopen hierdoor te bereiken dat de evaluatie realistischer is en bovendien door een grotere categorie gebruikers kan worden uitgevoerd.

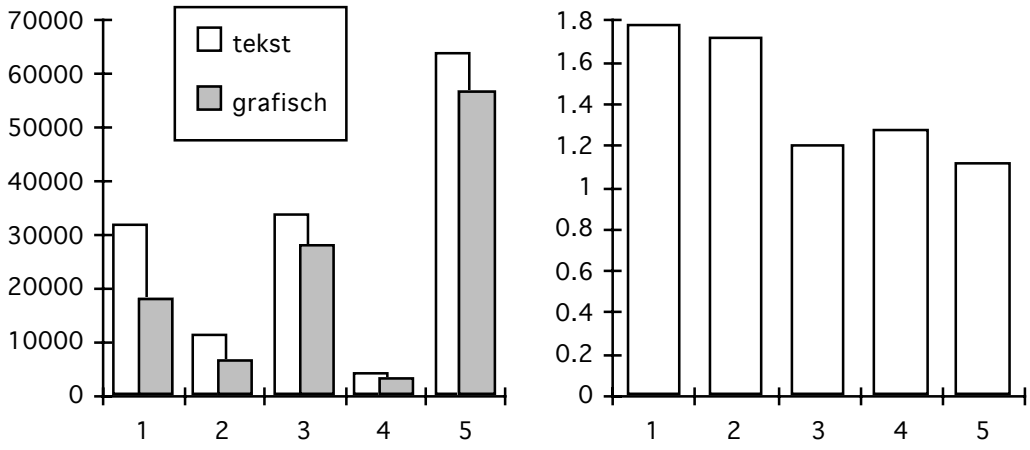
Objectieve evaluatie

Ten eerste kunnen we kijken naar de onderlinge verschillen voor de opdrachten afzonderlijk. Onderstaande figuur geeft grafieken te zien van de absolute en relatieve opdrachtduren† voor de tekstuele en grafische programmeermethode. We mogen aannemen dat de duur van de opdracht samenhangt met de ondervonden moeilijkheidsgraad (dus niet noodzakelijkerwijs hetzelfde als de inherente complexiteit).

Wat allereerst opvalt is dat de verschillen, vooral voor de langere opdrachten, absoluut gezien nogal klein zijn. Dit blijkt vooral uit de relatieve opdrachtduur van opdrachten 3, 4, en 5, die daar bijna gelijk zijn aan 1. Het lijkt dat vooral in het begin het grafische interface relatief duidelijk sneller was dan het tekstuele.

---

† Al onze tijdmetingen zijn gedaan in *ticks*, d.w.z. 1/60 van een seconde

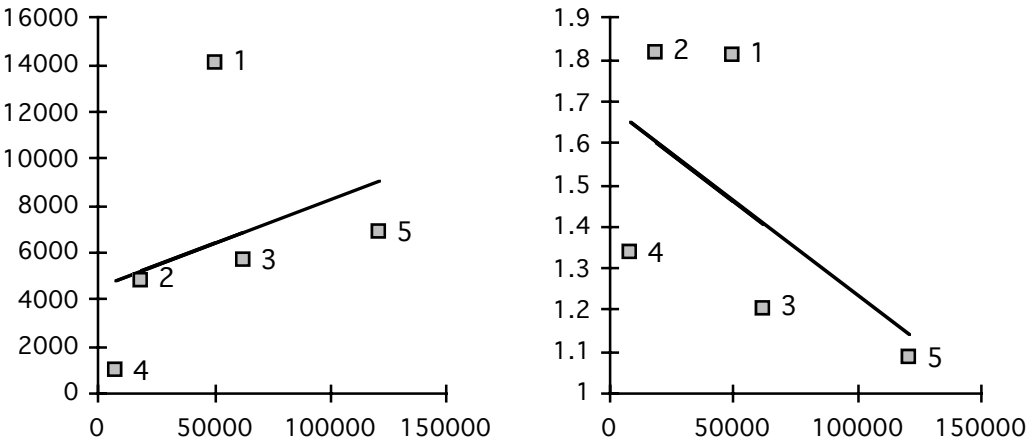


Figuur 25. Absolute (links) en relatieve (rechts) duur van een tekstuele/grafische opdracht, naar opdracht

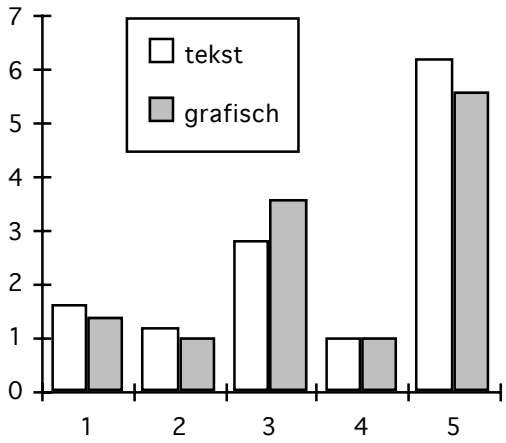
We kunnen het verband tussen de opdrachtduur (ondervonden opdrachtcomplexiteit) nog wat explicieter naar voren brengen met behulp van de twee grafieken in de volgende figuur. Deze geven respectievelijk de absolute en relatieve verschillen aan tussen de duur van een tekstuele en grafische opdracht, naar de gemiddelde absolute lengte van de opdracht. Om de interpretatie enigszins te vereenvoudigen hebben we de lineaire interpolatie van het verschil volgens de methode van de kleinste kwadraten in de grafiek uitgezet. Dit geeft slechts een *trend* in de relatie aan en is niet geschikt voor nadere quantitatieve interpretatie.

We zien hier duidelijk dat naarmate de opdrachten meer tijd vergen het grafische interface sneller tot resultaten leidt dan het tekstuele. Als we echter beschouwen hoe groot dit verschil is vergeleken met de duur van de gehele opdracht, dan zien we dat het relatieve voordeel van het grafische interface voor complexere opdrachten eigenlijk juist afneemt.

Het tegenvallende resultaat, dat de grafische programmmeertests bij de meer



Figuur 26. Verschil in duur (links) en relatieve duur (rechts) van de opdrachten, naar gemiddelde duur



Figuur 27. Gemiddeld aantal compilaties, naar opdracht

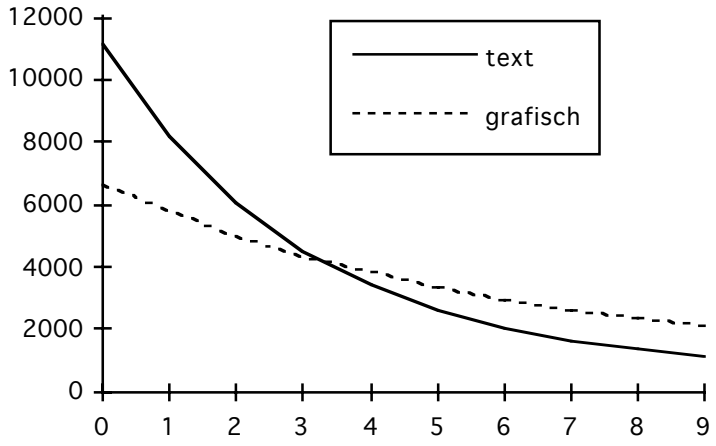
ingewikkelde opdrachten niet de verwachte verlaging van de benodigde tijd met zich mee brachten, zou onder meer kunnen worden verklaard met de observatie dat de meeste gebruikers moeilijkheden hadden met het verkrijgen en behouden van een goed structureel overzicht van de complexere programma's in de grafische tests, en met het daarmee verbonden herrangschikken van de commando's wanneer een fout gemaakt werd.

Om de oorzaak hiervan te proberen achterhalen kijken we naar het gemiddeld benodigde aantal compilaties per opdracht. Aan de hand van figuur 27 kunnen we zien dat de gebruikers met opdrachten 3 en 5 meer moeite hadden, zoals te verwachten was. In figuur 26 zien we inderdaad dat bij met name deze twee opdrachten het voordeel van het grafische interface in verhouding tot het tekstuele teniet gedaan wordt. De observatie dat ook bij opdracht 4 het grafische interface relatief weinig winst oplevert, terwijl dit zeker geen complexe opdracht was, wordt waarschijnlijk veroorzaakt doordat de gebruikers voor een opdracht van deze lage complexiteit het punt al beginnen te naderen waarop tekstuele invoer sneller wordt dan grafische invoer (bedenk hierbij dat de referentieopdrachten 2 en 4 bijna identiek zijn). Dit *cross-over effect* wordt verderop nader onderzocht.

De vraag is nu waarom een gebruiker als hij een grafisch interface gebruikt relatief meer moeite heeft met complexe opdrachten? Uit observaties van de gebruikers kunnen we als verklaring aanvoeren dat gebruikers veel moeite hebben met zogenaamde hoog-niveau edit-acties in het grafische interface. Hieronder verstaan we bijvoorbeeld het herrangschikken van commando's. Het aantal benodigde compilaties weergeven in figuur 27 is te beschouwen als een maat voor het aantal uitgevoerde edit-acties.

Het volgende probleem bleek zich bijvoorbeeld voor te doen. Het automatische grid dat we hadden geïmplementeerd om de gebruiker te ondersteunen bij het overzichtelijk houden van het programmeerveld bleek in de praktijk een tegengesteld effect te hebben, namelijk dat gebruikers verrast werden door het plotselinge verschuiven van de commandoiconen, met als gevolg dat vaak meerdere iconen in het grid over elkaar heen kwamen te liggen. We hadden hier zonder ons dat te realiseren het eerder gevonden user-interface principe van “user-control” genegeerd, en het resultaat daarvan is ons duidelijk merkbaar geworden.

Een ander veelvoorkomend probleem betreft het toevoegen en verwijderen van commando's. Het was namelijk uit de documentatie niet duidelijk op welke plaats nieuwe commandoiconen in het programma ingevoegd werden, noch repte de instructie enig woord over de procedure waarmee iconen uit het programma verwijderd konden worden.



Figuur 28. 'Leercurves' voor tekstuele en grafische interfaces

We kunnen concluderen dat de Structure Manager nog aanzienlijk verbeterd zou kunnen worden zodat hoog-niveau edit-acties makkelijker uit te kunnen voeren. Het gebruik van een librarysysteem zoals de voorgestelde Library Manager (hoofdstuk "Applicatie") lijkt veel van dit soort problemen te kunnen voorkomen, doordat de hoog-niveau commandostructuren hierin kunnen worden opgeslagen en de gebruiker slechts de parameters van de commando's hoeft aan te passen.

Vervolgens kijken we naar de exponentiële leerfuncties. Hierbij modeleren we het leerproces als een afname van de tijd die nodig is voor de uitvoering van een bepaalde opdracht, volgens een eerste-orde exponentiële functie

$$t = t_{\infty} + \mu e^{-\lambda x}$$

Hierin is x het verloop van tijd (in ons geval gelijk gesteld aan de hoeveelste keer een gebruiker de opdracht uitvoert, d.w.z. 0 en 1 voor respectievelijk de eerste en de tweede keer) en t de duur van de opdracht. t_{∞} is de 'convergente eindduur', d.w.z. de tijd waarin een oneindig ervaren gebruiker de opdracht zou uitvoeren. Hiervoor hebben we de tijdsduur genomen die wij zelf voor de opdracht nodig hadden, zijnde $t_{\infty tekst} = 614$ en $t_{\infty grafisch} = 799$.

We vinden $\lambda_{tekst} = 0.33$ ($\sigma = 0.32$) met $\mu_{tekst} = 10517$ ($\sigma = 5869$), en $\lambda_{grafisch} = 0.17$ ($\sigma = 0.35$) met $\mu_{grafisch} = 11520$ ($\sigma = 7261$). We kunnen dit resultaat ook beschrijven door te zeggen dat onze testgebruikers gemiddeld twee keer zo snel 'leerden' gebruikmakend van het tekstueel interface [sic], maar dat komt omdat ze ook met een lagere snelheid begonnen ($\mu_{tekst} > \mu_{grafisch}$). Als we deze twee 'leercurves' tegelijk in een grafiek uitzetten krijgen we het volgende:

We zien hier rond de derde opdracht inderdaad een 'cross-over' punt, waarbij het grafisch interface zijn relatieve snelheid verliest ten gunste van het tekstuele. Het verbaast ons niet dat er zo'n cross-over punt bestaat: het is in overeenstemming met de ervaring van veel gebruikers dat wanneer men eenmaal erg ervaren is het grafische interface een zekere overhead inhoudt. Overigens bleef ook voor deze groep van gebruikers gelden dat ze het grafische interface niettemin plezieriger vonden werken.

Het kleine aantal subjecten maakt onze conclusies statistisch slecht betrouwbaar. We hebben echter geen keuze maar het is duidelijk dat het eigenlijk nodig zou zijn om de tests met veel meer gebruikers te herhalen.

Subjectieve evaluatie

De resultaten van deze test verrasten ons zeer. De meeste gebruikers, behalve zeer ervaren programmeurs, bleken grote moeite te hebben met het uitvoeren van de tests. Met name bleek de door ons zeer eenvoudig veronderstelde Turtle taal zelf zeer aanzienlijke problemen op te leveren. De door ons bewust ingebouwde moeilijkheden zoals de verschillende coordinaatstelsels bleken soms haast onoverkoombare barriëres. Interresant was het hierbij op te merken dat meer ervaren programmeurs tijdens het lezen vaak al een aantal van deze problemen zagen en er aan de begeleiders vragen over stelden. Kennelijk zijn zij al ingesteld op dit soort problemen.

We hoopten in de uitvoering van de tests verschillende werkstijlen waar te kunnen nemen. Hierbij valt te denken aan veel debugcycli waarbij het programma incrementeel wordt verbeterd, tegenover een programmeerstijl waarbij het programma in één keer zorgvuldig in zijn geheel wordt geschreven. Het bleek dat zowel ervaren programmeurs als nieuwelingen probeerden in één keer het gehele programma te schrijven. Bij navraag werd door de ervaren gebruikers gezegd dat ze dachten dat het probleem zó simpel was dat ze het wel in een keer konden oplossen. Dit viel overigens in de praktijk vaak tegen. De onervaren gebruikers bleken er bij navraag niet aan gedacht te hebben dat het wel eens makkelijker zou kunnen zijn om twee keer vijf regels te debuggen dan het is om in een keer tien regels te debuggen. Slechts één gebruiker was hier duidelijk een uitzondering op en deelde het moeilijkste probleem op in twee delen die hij na elkaar debugde.

Gebruikers lieten zich duidelijk positiever uit over het grafische interface. Bij het tekst interface klaagden de meesten al snel dat ze weer "`START program`" moesten intypen. Bij het grafische interface klaagde niemand, zelfs niet toen een gebruiker anderhalf uur (!) bezig geweest was met vijf opdrachtjes. Het heeft ons vaak verbaasd hoe volhoudend de testgebruikers soms waren. Als ze aangeboden werd om te stoppen met de test omdat het niet erg wilde lukken waren ze eigenlijk niet tegen te houden, en wilden ze koste wat koste de opdracht succesvol afmaken (waarvoor wij ze uiteraard zeer dankbaar zijn).

De instructies die we bij de opdrachtomschrijvingen leverden waren toegespitst op de tekstuele opdrachten en in mindere mate geschikt voor de grafische tests. Het bleek tevens dat men bij de tekstuele opdrachten veel vaker toevlucht moest nemen tot de documentatie. De voor de grafische tests ietwat beperkte documentatie leverde echter in het algemeen heel weinig problemen op, al hadden de onderstaande problemen voorkomen kunnen worden als we een klein voorbeeld hadden gegeven van de grafische specificatie van de commando's. Nadeel hiervan is echter wel dat de gebruikers nog meer moeten lezen. Het bleek dat de meeste gebruikers met tegenzin de instructies lazen en ze bovendien vaak erg slecht lazen. Dit is in overeenstemming met [Fleischacker90] waar dit ook al geconstateerd wordt. Het is ons duidelijk dat een dikker pak papier voor de instructie voor veel gebruikers te intimiderend zou zijn.

De door ons gekozen defaultwaarden in de dialogen bleken voor veel testgebruikers een probleem. Het was niet intuïtief duidelijk dat als er een "y" stond ze deze moesten vervangen door een getal. Achteraf gezien was het waarschijnlijk beter geweest om hier bijvoorbeeld een geschikte 'legale' waarde als default te gebruiken, zoals "0".

Ook de 'color picker', die gebruikt wordt in de grafische programmeertest om kleuren te selecteren, had enige uitleg nodig voordat de gebruikers begrepen hoe hij werkte, terwijl het gebruik hiervan ons toch vanzelfsprekend leek.

De manipulatie van de iconen leverde ook meer problemen op dan we hadden verwacht. Zoals we al eerder geconstateerd hebben, lijkt het ons nuttig om nog meer na te denken over hoe de Structure Manager, die verantwoordelijk is voor de icoon manipulaties, verbeterd kan worden.

Conclusies

Het hier beschreven onderzoek heeft geleid tot resultaten die betrekking hebben op het gebruik van grafische gebruikersinterfaces. Een aantal onderwerpen die van bijzonder belang waren voor het TRIP user-interface zijn specifiek onderzocht— hieronder vallen de iconische representatie en de invoer van een specificatie met behulp van zowel een commando- als een grafische taal.

De gebruikte methode voor de evaluatie van iconen is zeer nuttig gebleken. Er is meer inzicht verkregen in de denkwijzen van typische gebruikers en daarmee ook in welke manier van iconische representatie het meest geschikt is voor onze categorie van toepassingen. Het blijkt dat het gebruik van iconen die volgens de directe methode ontworpen zijn tot de kortste selectietijd leiden, en bovendien voor de gebruikers goed te herkennen zijn. Dit laatste is met name nuttig voor nieuwe gebruikers die snel de associatie tussen de uit te voeren actie (het commando) en het icoon kunnen vinden. Het blijkt dat het gebruik van kleur niet echt bijdraagt aan de herkenning van de iconen maar wel de selectiesnelheid bevordert. Om tot statistisch sterkere uitspraken te mogen overgaan is een aanzienlijk groter aantal testgebruikers noodzakelijk. Dit is echter een probleem gezien de beperkte tijd die beschikbaar is voor het uitvoeren van deze opdracht.

In onze resultaten hebben we bovendien een bevestiging gevonden van in de literatuur beschreven suggesties voor het ontwerpen van iconen. Hieronder valt bijvoorbeeld het verwijderen van onnodige details uit de ontwerpen.

Bij het schrijven van toekomstige programma's zullen wij beiden zeker gebruik kunnen maken van de gebruikte ontwerpmethodologieën en software om iconen te evalueren. Bovendien zijn we ons veel beter bewust geworden van de problemen die kunnen optreden bij, met name, het herkennen van iconen door onervaren gebruikers.

De resultaten van ons onderzoek laten zien dat een grafisch gebruikersinterface voor beginnende gebruikers een snelheidswinst oplevert. Voor alle gebruikers geldt dat ze

het grafische interface plezieriger vonden om mee te werken, alhoewel blijkt dat voor ervaren gebruikers het tekstuele interface sneller werkt. We denken dat het grafische interface nog aanzienlijk verbeterd kan worden zodat hij optimaler werkt voor een grotere groep van gebruikers en categorie opdrachten. Het bijkt echter wel dat met de textuele interface sommige gebruikers de opdrachten niet konden uitvoeren dit bij het grafische interface niet voorkwam. Het grafische interface werkt duidelijk drempel verlagend.

We hadden duidelijk onderschat hoeveel achtergrondinformatie er nodig is om te kunnen programmeren. Het gaat hierbij om bepaalde basiskennis, zoals het gebruik van coordinaatstelsels, windows en dialogen, clipboardoperaties, e.d.

Helaas hebben we zelf gedurende de afgelopen jaren op de Universiteit Twente moeten constateren dat 'universitaire' software vaak bij lange na niet voldoet aan de minimum-eisen die aan gebruikersvriendelijke software gesteld mogen worden. Dit is jammer en kortzichtig omdat het de leertijd vergroot en het toepassingsdomein onnodig verkleint. Het lijkt ons van groot belang dat in het programmeeronderwijs zoals dat nu gegeven wordt meer aandacht aan dit onderwerp besteed wordt.

Het is ons ook duidelijk geworden dat het ontwerpen van tamelijk complexe software aan een universiteit uiterst problematisch is, vooral als deze software onderdelen bevat die niet *meteen* aansluiten op het aldaar uitgevoerde onderzoek. Dit komt niet in de laatste plaats door het gebrek aan continuïteit door de relatief korte werkverbanden van promovendi en studenten. Dit soort projecten kunnen alleen succesvol uitgevoerd worden indien de vaste staf bereid is zich er voor in te zetten.

Tot slot is het misschien aardig om op te merken dat er ook vanuit de industrie belangstelling is voor de gebruikte methode. Dit maakt het des te spijtiger dat de programmatuur nooit is gecompleteerd.

FUTURE WORK

Op twee gebieden blijft nog veel werk te doen. Wat betreft de software is dit met name het completeren van een implementatie die onder X Window en OSF/Motif kan draaien. Maar ook aan de kant van de meer formele informatica is nog veel te doen. Hierbij denken we met name aan het nauwkeuriger specificeren van de benodigde taaltransformaties. Hierover zijn in de respectievelijke verslagen meer suggesties te vinden.

Op het gebied van de ergonomie zijn helaas ook nog een groot aantal vragen blijven staan. Deze lopen uiteen van de evaluatie van de door ons gebruikte dialooglayout generatiealgorithmen tot het nauwkeuriger onderzoeken en vergelijken van de leerprocessen bij zowel de op tekst gebaseerde gebruikersinterfaces en de grafische variant daarvan.

Ook blijft het punt dat onze resultaten vaak moeilijk statistisch significant te maken zijn omdat daarvoor veel meer testgebruikers nodig zijn. Er is echter aangetoond dat het uitvoeren van een formele evaluatie met behulp van testgebruikers mogelijk is. Uiteraard zou een uitbreiding naar een zeer groot aantal testgebruikers interresant zijn maar dit is binnen de opzet van dit vak niet mogelijk.

Appendices

Appendix A geeft het relevante deel van de `lex` syntaxspecificatie en appendix B dat van de `yacc` grammaticaspecificatie voor de turtle simulatortaal. In appendix C worden de instructies en de grafische en tekstuele programmeertests beschreven

APPENDIX A — SYNTAX

```
%{
/*
        turtle.l

        lexical analyzer source file
        for turtle, the TRIP demonstration language

        Copyright © 1992 by: Ben Hekster
*/
%}

cr                      [\n]
delim                   [ \t]
letter                  [A-Za-z]
digit                   [0-9]

integer   [+-]?{digit}+


%%

{cr}                    { yylineno++; return CR; }
{delim}+                { /* whitespace */ }

START                   { return START; }
END                     { return END; }
program                 { return PROGRAM; }

lineStyle:              { return LINESTYLE; }
plain                   { return PLAIN; }
thicker                 { return THICKER; }
thickest  { return THICKEST; }

COLOR                   { return COLOR; }
red:                    { return RED; }
green:                  { return GREEN; }
blue:                   { return BLUE; }

RECTANGLE               { return RECTANGLE; }
CIRCLE    { return CIRCLE; }
radius:                 { return RADIUS; }

deg                     { return DEG; }
rad                     { return RAD; }
grad                    { return GRAD; }

MOVE                    { return MOVE; }
from:                   { return FROM; }
to:                     { return TO; }

\,                      { return COMMA; }

{integer}               { yylval.literalValue = atoi(yytext); return INTEGER; }
```

APPENDIX B — GRAMMATICA

```
{%
/*
        turtle.y

        parser grammar specification
        for turtle, a TRIP demonstration language

        Copyright © 1992 by: Ben Hekster
*/
%}


%start          program

%token          START END PROGRAM
                LINESTYLE PLAIN THICKER THICKEST
                CIRCLE RECTANGLE
                RADIUS DEG RAD GRAD
                COLOR RED GREEN BLUE
                MOVE FROM TO
                COMMA CR
                INTEGER

%union {
                long            unitConversion;
                signed short    literalValue;
                }

%type <literalValue> INTEGER
%type <unitConversion> angleUnit


%%


/*
        turtle program
*/

program:
                startProgramCommand
                blockCommandList
                endProgramCommand
        ;

startProgramCommand:
                START PROGRAM CR
        ;

endProgramCommand:
                END PROGRAM CR
        ;

blockCommandList:
                blockCommandList blockCommand CR
        |
        ;

blockCommand:
                movePenCommand
        |
                rectangleCommand
        |
                circleCommand
        |
                colorCommand
        ;

movePenCommand:
                MOVE TO INTEGER COMMA INTEGER lineStyle
        ;

rectangleCommand:
```

```
                    RECTANGLE RADIUS INTEGER COMMA INTEGER lineStyle

circleCommand:
                    CIRCLE RADIUS INTEGER COMMA INTEGER
                    FROM INTEGER TO INTEGER angleUnit lineStyle

angleUnit:
                    DEG
          |
                    RAD
          |
                    GRAD
          ;

colorCommand:
                    COLOR RED INTEGER GREEN INTEGER BLUE INTEGER

lineStyle:
                    LINESTYLE thickness
          ;

thickness:
                    PLAIN
          |
                    THICKER
          |
                    THICKEST
          ;

%%
```

APPENDIX C — INSTRUCTIES PROGRAMMEER OPDRACHT

OPZET

In het kader van het college *Gebruikersvriendelijkheid van Geautomatiseerde Systemen* van de Universiteit Twente proberen wij te onderzoeken op welke manier de omgang van de gebruiker met een computer vereenvoudigd kan worden. Hiervoor hebben wij een aantal tests ontwikkeld, in de vorm van kleine opdrachten die U met behulp van de computer uitvoert.

Aan de hand van hoe deze opdrachten uitgevoerd worden hopen wij af te kunnen leiden welke aspecten van het computergebruik verbeterd kunnen worden en hoe dit kan gebeuren. Het is dus belangrijk dat U zich realiseert dat het zeer wel mogelijk is dat bepaalde opdrachten voor U moeilijk of zelfs niet uitvoerbaar blijken te zijn.

Verder willen wij U hierbij nogmaals verzoeken om bij het uitvoeren van de opdrachten 'hardop na te denken'. Door hardop te denken terwijl U de opdrachten uitvoert kunnen wij veel beter inzicht krijgen in de specifieke problemen in het computergebruik.

INSTRUCTIES

Op de computer gaat U zometeen een programma gebruiken waarmee U eenvoudige 'tekeningen' kunt maken. Deze tekeningen worden *niet* gemaakt door, bijvoorbeeld, met de muis lijnen te trekken— in plaats daarvan zult U steeds kleine 'programmaatjes' schrijven om deze tekeningen te maken.

U kunt tekenen door een schildpad, waaraan een pen is bevestigd over een groot vel papier beweegt. Door de schildpad te laten bewegen laat de pen een 'spoor' achter op het papier. U kunt dus tekeningen maken door de schildpad de juiste opdrachten (zogenaamde *commando's*) te geven. Deze *serie* van commando's vormen het 'programma'.

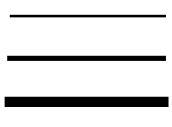
U moet steeds een heel programma tegelijk aan de schildpad geven. U kunt dus niet de commando's één voor één doorgeven. De schildpad begint in het midden van het papier, voert het programma uit, en keert naar het beginpunt terug. Als blijkt dat de dan onstane tekening niet de gewenste is, kunt U het programma aanpassen en vervolgens opnieuw door de schildpad laten uitvoeren. U kunt dit net zo vaak herhalen totdat de resulterende tekening correct is.

HET PROGRAMMEREN

Er zijn twee verschillende manieren waarop U commando's aan de schildpad zult gaan geven:

- U kunt de commando's intypen, in een voor de schildpad begrijpbare taal. De schildpad stelt echter strikte eisen aan de correctheid van de commando's, zodat U deze zeer nauwkeurig zult moeten formuleren. Deze manier van programeren wordt onderzocht in de textuele programmeertests.
- U kunt ook de commando's door de computer zelf laten maken. U moet dan uit een menu het geschikte commando kiezen. De computer vraagt U dan om de benodigde gegevens, en genereert dan zelf het commando zodat U het zelf niet hoeft in te typen. In plaats van het commando ziet U op het scherm een icoon. Het hele programma wordt dus een keten van iconen. In de grafische programmeertests wordt deze manier van programmeren onderzocht.

De begeleider zal U vertellen welke opdrachten en in welke volgorde U ze moet uitvoeren.



lijnsoort is “plain”

lijnsoort is “thicker”

lijnsoort is “thickest”

SCHILDPAD PROGRAMMA'S

Een correct ‘schildpad programma’ moet op de volgende manier zijn opgebouwd:

- Het programma moet beginnen met een ‘start’ commando
- Het programma moet eindigen met een ‘end’ commando
- Hiertussen mogen zich een willekeurig aantal commando's bevinden. De commando's mogen in een willekeurige volgorde staan. De volgende commando's mogen gebruikt worden: ‘move, ‘rectangle’, ‘circle’, en ‘color’. De betekenis van deze commando's wordt in het volgende deel uitgelegd

U kunt een programma aan de schildpad aanbieden door “Compile” uit het “File” menu te kiezen. Normaal gesproken krijgt U dan (na enige vertraging, aangezien de schildpad tijd nodig heeft om Uw instructies uit te voeren) in het “Turtle Output” window het resultaat van uw programma te zien in de vorm van een tekening.

Echter, wanneer uw programma onjuist is opgebouwd, bijvoorbeeld doordat een benodigd commando ontbreekt, of omdat een woord in een commando verkeerd is gespeld, of een commando onjuiste informatie bevat, stopt de schildpad met het uitvoeren van Uw commando's en schrijft de tekst “parse error”. U moet dan proberen het commando te vinden waar de fout zit, en de fout corrigeren. U kunt dan het verbeterde programma opnieuw aan de schildpad aanbieden.

SCHILDPAD COMMANDO'S

Hieronder volgt een opsomming van de beschrijvingen van commando's die de schildpad begrijpt. Elk commando moet op een nieuwe regel beginnen (U kunt een nieuwe regel beginnen met de “return” toets). Let op dat U de hoofd- en kleine letters precies moet overnemen! Alle commando's moeten met een return worden afgesloten. In de grafische programmeertests neemt de computer een groot deel van de commando specificatie op zich zodat U niet alles zelf hoeft in te typen.

Als in de commandobeschrijvingen een *schuin gedrukt* woord voorkomt, dan betekent dit dat U hiervoor iets anders moet invullen. Bijvoorbeeld:

lijnsoort Wanneer dit in een commando voorkomt, moet U één van de volgende woorden invoeren: “plain”, “thicker”, of “thickest”. Deze geven elk een verschillende dikte aan van de pen waarmee de schildpad tekent (zie figuur):

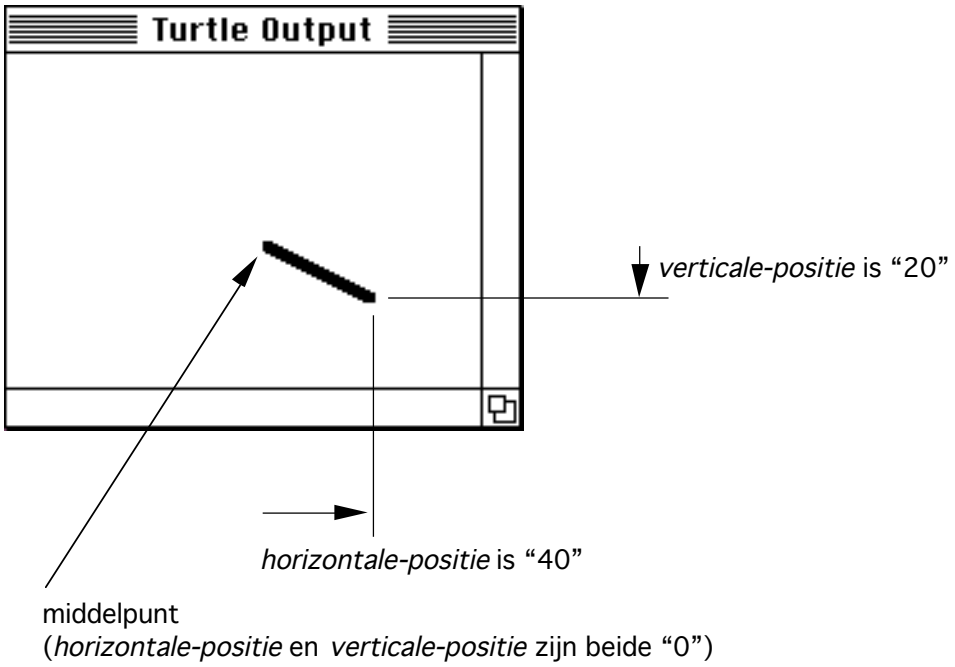
START program

Begint het schildpad programma. De schildpad gaat naar het midden van de “Turtle Output” window.

END program

Sluit een schildpad programma af. De schildpad stopt met tekenen.

MOVE to: *horizontale-positie*, *verticale-positie* lineStyle: *lijnsoort*



Vertelt de schildpad om in een rechte lijn naar een nieuwe plaats te gaan. Deze nieuwe plaats geeft U aan door de *horizontale-positie* en *verticale-positie* vanaf het middelpunt van het blad papier.

Voor *horizontale-positie* moet U een getal invoeren dat de horizontale positie vanaf het middelpunt van het “Turtle Output” window aangeeft (zie figuur). Merk op dat positieve getallen (bijv. “40”) posities *rechts*, en negatieve getallen (bijv. “–40”) posities *links* van het middelpunt voorstellen.

Evenzo moet U voor*verticale-positie* een getal invoeren dat de verticale positie vanaf het middelpunt van het “Turtle Output” window aangeeft (zie figuur). Merk op dat positieve getallen (bijv. “20”) posities *onder*, en negatieve getallen (bijv. “–20”) posities *boven* het middelpunt voorstellen.

De dikte van de pen waarin de schildpad tekent kunt U aangeven door *lijnsoort*.

RECTANGLE radius: *horizontaal*, *vertikaal* lineStyle: *lijnsoort*

Vertelt de schildpad om een rechthoek te tekenen gecentreerd rondom het punt waar hij nu is. De breedtestraal van de rechthoek kunt U aangeven door *horizontaal*, en de hoogtestraal door *vertikaal* .

De dikte van de pen waarin de schildpad tekent kunt U aangeven door *lijnsoort*.

CIRCLE radius: *horizontaal*, *vertikaal* from: *beginhoek* to: *eindhoek hoek-eenheid* lineStyle: *lijnsoort*

Deze opdracht vertelt de schildpad om een (deel vaneen) cirkel of ellips te tekenen gecentreerd rondom het punt waar hij nu is. De breedte van de cirkel of ellips kunt U aangeven door *horizontaal*, en de hoogte door en getal voor *vertikaal te substitueren*.

De cirkel wordt getekend vanaf de hoek aangegeven door *beginhoek* tot en met de hoek aangegeven door *eindhoek*. De eenheid waarin U deze beide hoeken specificeerd kunt U aangeven door *hoek-eenheid*: deze is “deg” voor graden, “grad” voor gradialen, en “rad” voor radialen. Een volledige cirkel loopt van 0 deg tot 360 deg, of 0 grad tot 400 grad, of 0 rad tot 2π rad. Het 0 deg punt bevindt zich rechtboven het centrum van de cirkel.

De dikte van de pen waarin de schildpad tekent kunt U aangeven door *lijnsoort*.

COLOR red: *rood-heid* green: *groen-heid* blue: *blauw-heid*

Deze opdracht vertelt de schildpad om vanaf dat punt met een nieuwe kleur te gaan tekenen. De kleur geeft U aan door middel van de hoeveelheid rood, groen, en blauw in de kleur. De uiteindelijke kleur ontstaat door het mengen van deze drie componenten. De *rood-heid*, *groen-heid*, en *blauw-heid* zijn getallen van 0 tot en met 65 535.

Bijvoorbeeld, een zuivere kleur rood heeft een *rood-heid* van 65 535, en een *groen-heid* en *blauw-heid* van 0. Andere kleuren ontstaan door rood, groen, en blauw te mengen.

Bij de grafische programmeertests kunt U een kleur kiezen uit het getoonde palet door deze direct aan te klikken met de muis.

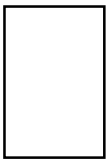
OPDRACHTEN

Er volgt nu een serie van tien kleine tekeningen, die U één voor één door de schildpad moet laten tekenen. De eerste 5 moet U tekenen met behulp van een programma dat U met behulp van tekstmethode invoert. De tweede serie van vijf opdrachten moeten met behulp van het grafische gebruikersinterface ingevoerd worden.

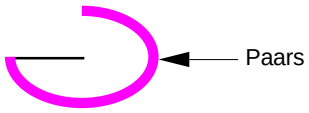
De eerste serie van vijf opdrachten begint hier. Het is de bedoeling dat ze met behulp van de tekstueleprogrammeer methode uitgevoerd worden. Voor U begint laat Uw begeleider zien hoe U een commando kan invoeren en verbeteren.

Opdracht 1t

Opdracht 2t

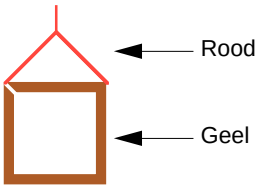


Opdracht 3t



Opdracht 4t

Opdracht 5t

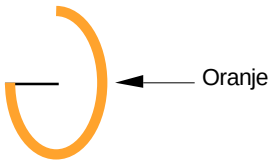


De tweede serie van vijf opdrachten begint hier. Het is de bedoeling dat ze met behulp van de grafische programmeermethode uitgevoerd worden. Voor U begint laat Uw begeleider zien hoe U een commando kan invoeren en verbeteren.

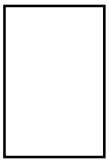
Opdracht 1g

Opdracht 2g

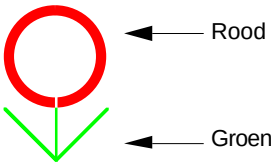
Opdracht 3g



Opdracht 4g



Opdracht 5g



Referenties

[Amstel88] J.J. Van Amstel, J.A.A.M. Poirters, *Voortgezet programmeren: Het Ontwerpen van Datastructuren en Algoritmen*, Academic Service, 3e druk 1988

[Anonymous90] *23 Windows "Pains"*, AppleLink Student Representative folder, 1990

[Apple85] Apple Computer, *Inside Macintosh*, vol. I-VI, Addison-Wesley 1985, 1986, 1988, 1991

[Apple86] Apple Computer, *Human Interface Guidelines: The Apple Desktop Interface,* Addison-Wesley 1986

[Apple88a] Annette Wagner, Human Interface Group internal memorandum, Apple Computer, 02/06/88

[Apple88b] Annette Wagner, Human Interface Group internal memorandum, Apple Computer, 13/06/88

[Apple88c] Kathleen Gomoll en Anne Nicol, *Human Interface Update #11*, Apple Computer, 25/04/88

[Apple89] Apple Computer, *Human Interface Notes*, HyperCard stack

[Bakker89] P. Bakker, *Opdracht Gebruikersvriendelijkheid van Geautomatiseerde Systemen I & II,* schriftelijke communicatie, september 1989

[Bell] Bell Telephone Laboratories, *UNIX Programmer's Manual*, Murray Hill, New Jersey

[Buurman85] Buurman, et al., *beeldschermergonomie; Beeldschermwerk, Ergonomische achtergronden, Aanbevelingen*, Nederlandse vereniging voor ergonomie, 1985

[Efe87] Kemal Efe, *A Proposed Solution to the Problem of Levels in Error-Message Generation*, p. 948, Comm. ACM (30) 11, november 1987

[EG91] *Werken met beeldschermen*, Europese richtlijn voor het werken met beeldschermen, 1991

[Fleischhacker90] L.E. Fleischhacker, H. Alblas, M. Steenhouder, B. Ziegler-Jung, *Mens en Informatietechniek*, Universiteit Twente, 1990

[Foley82] James D. Foley en Andries van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley 1982

[Foley90] James D. Foley, Andries van Dam, Steven K. Feiner en John F. Hughes, *Computer Graphics: Principles and Practice*, Second Edition, Addison-Wesley 1990

[Hacker91] *The New Hacker's Dictionary*, MIT Press, Cambridge MA, 1991.

[Hall88] F.E. Hall en J.B. Byers, *X: A Window System for Distributed Computing Environments*, p. 46, Hewlett-Packard Journal (39) 5, oktober 1988

[Hekster90] Ben Hekster, trip: A Flexible Input Processor for trendy, Phase II: Design and Implementation of a Syntax-Directed Hierarchical Dynamic Dialog System, verslag 250-uurs opdracht, juni 1990

[HP87] Hewlett-Packard, *Programming with the X Window System: HP 9000 Series 800 and Series 300 Computers*, 1987

[HP89a] Hewlett-Packard, *Using the X Window System 1*, 1989

[HP89b] Hewlett-Packard, *Using the X Window System 2*, 1989

[HP89c] Hewlett-Packard, *A Beginner's Guide to the X Window System*, second edition, september 1989

[Kalsbeek79] J.W.H. Kalsbeek, F.W. Umbach, *Het werken met beeldschermen*, Kluwer, Deventer, 1979

[Marcus91] Aaron Marcus, Andries van Dam, User-Interface Developments for the Nineties, IEEE Computer, blz. 49-57, september 1991

[Middelhoek89a] P.F.A. Middelhoek, *Design and Partial Implementation of TRIP , a Flexible Input Processor for TRENDY* , verslag 250-uurs opdracht, october 1989

[Middelhoek89b] P.F.A. Middelhoek, *Design of a User Interface for the CARAD Program, verslag 100-uurs opdracht*, februari 1989

[OSF90a] Open Software Foundation,OSF/Motif™ User's Guide, Prentice-Hall, Inc., 1990

[OSF90b] Open Software Foundation, OSF/Motif™ Style Guide, Prentice-Hall, Inc., 1990

[Remmers90] R. Remmers, *Spreading Resistance and SIMS Profiling of Silicon Structures*, verslag 250-uurs opdracht, juli 1990

[Schie90] Eddie van Schie, *TRENDY: An Integrated Program for IC process and device simulation*, proefschrift, april 1990

[Sloff77] J.J. Sloff, A. Yntema, G. Krooshof en T.G.H. Aalders, *Mathematische Statistiek*, Wolters-Noordhoff, 1977

[Smit87] J.H.A. de Smit en E.A. van Doorn, *Kansrekening voor EL*, dictaat, juni 1987

[Verbeek89] L.A.M. Verbeek, *Inleiding in de theoretische informatika*, Universiteit Twente, February 1989

[Weiser91] Mark Weiser, *The Computer for the 21st Century*, Scientific A merican, september 1991, blz. 66

[Wolbert91] Philip Wolbert, *Modeling and Simulation of Semiconductor Devices in TRENDY: Electrical, Thermal and Hydrodynamic Behavior*, proefschrift, oktober 1991