Help... de informatie-analist verzuipt!

HELP ... DE INFORMATIE-ANALIST VERZUIPT!

door ir. J. G. de Boer en ir. R. Vonk

Methodieken die gekenmerkt kunnen worden als representanten van de zogenaamde procesbenadering maken – een enkele uitzondering daargelaten – bij het opstellen van systeemspecificaties gebruik van grafische beschrijvingswijzen (zoals A- en I-schema's in ISAC) en een top-down aanpak. In bepaalde situaties heeft een procesbenadering om een aantal redenen duidelijk de voorkeur boven een gegevensbenadering. Het onderhouden, en bewaken van de consistentie en volledigheid van de grote – en in dit top-down proces explosief groeiende – hoeveelheden analyse- en ontwerpgegevens vergt echter onevenredig veel tijd van de informatie-analist, aangezien deze systeemspecificaties in de (her)ontwerpfase voortdurend aan wijzigingen onderhevig zijn. Een geautomatiseerd hulpmiddel zou hier uitkomst kunnen bieden en het juiste gebruik van dit soort methodieken beter kunnen afdwingen.
Dit artikel start met een algemene beschouwing over de rol die functionele specificaties spelen in het informatiesysteemontwikkelingsproces en de eisen die aan geautomatiseerde hulpmiddelen gesteld dienen te worden. Vervolgens worden na een globale inventarisatie van bestaande geautomatiseerde hulpmiddelen de resultaten van een onderzoek, waarin een prototype van een dergelijk hulpmiddel ontworpen en gedeeltelijk gerealiseerd is, beschreven. Er is gekozen voor realisatie op een micro-computer, zodat de informatie-analist dit gereedschap met weinig moeite mee kan nemen naar de opdrachtgever. De reacties tot dusverre zijn zeer bemoedigend, en maken duidelijk dat een dergelijk systeem een oplossing kan bieden voor een door velen ervaren praktijkprobleem.

1 INLEIDING

Methoden en technieken voor het ontwikkelen en beheren van geautomatiseerde informatiesystemen in organisaties mogen zich heden ten dage in een nog immer stijgende belangstelling verheugen [Boer84]. Deze belangstelling stoelt vooral op de waargenomen problematiek rond het automatiseringsgebeuren en de informatievoorziening. Ontwikkelde informatiesystemen blijken niet effectief te zijn (beantwoorden niet aan de verwachtingen van de eindgebruiker), zijn slecht gestructureerd en kunnen slechts met veel pijn en moeite – of in het geheel niet – geïntegreerd worden met andere separaat ontwikkelde informatiesystemen. Het onderhouden van bestaande systemen vergt een onevenredig groot deel van de in de organisatie aanwezige automatiseringskapaciteit, en men ziet dan ook het verschijnsel dat de automatiseringsafdeling nauwelijks meer aan het ontwikkelen van nieuwe applicaties toekomt. Hierdoor groeit de ontevredenheid van de op automatiseringsgebied steeds mondiger wordende eindgebruiker.
Een groot gedeelte van deze problematiek wordt veroorzaakt door de nog steeds te geringe aandacht voor de eerdere fasen van het informatiesysteemontwikkelingstraject. De in de functionele ontwerpfase geïntroduceerde fouten, onvolkomenheden en onvolledigheden, vaak ontstaan door een ontoereikende communicatie tussen informatie-analist en (toekomstige) gebruiker van het in ontwikkeling zijnde informatiesysteem, leiden tot vaak zeer hoge kosten, een te lange projectrealisatietijd, en een zeer hoge 'onderhouds'-inspanning. Het verhogen van de kwaliteit van het functioneel ontwerp dient gezien het voorafgaande dan ook een hoge prioriteit te hebben bij diegenen die verantwoordelijk zijn voor de ontwikkeling van informatiesystemen.
Bij het zoeken naar meer inzicht in deze problematiek en naar oplossingen ervoor kan men in principe een tweetal elkaar niet uitsluitende wegen bewandelen. De ene weg kan het best worden getypeerd met het adjectief 'methodologisch', de andere met het adjectief 'technologisch'. De methodologische richting tracht via studie van werkmethoden in relatie tot typologieën van organisaties en van toepassingen tot meer inzicht in de materie te komen en van daaruit via verbeterde methodieken het hoofd te bieden aan de praktijkproblemen. De technologische richting tracht een bijdrage te leveren door zich te concentreren op de ondersteuning van de diverse te onderkennen ontwikkelingstaken en aan automatisering van de werkzaamheden.
Geavanceerde ontwikkelingshulpmiddelen, zoals vierde generatie talen maken bijv. prototyping (het op een iteratieve wijze bouwen, met de gebruiker evalueren, en wijzigen van een informatiesysteem, totdat een bevredigend resultaat is ontstaan) economisch haalbaar. Gehanteerd in een goed kader – iets dat in de praktijk nog wel eens wil ontbreken – is prototyping een veelbelovende aanpak. Niet elk informatiesysteem kan echter op een dergelijke wijze ontwikkeld worden [Gremi83]. Zo zal men in ieder geval een 'traditionele' methodiek moeten hanteren wanneer er sprake is van complexe, omvangrijke systemen met een lange levensduur, wanneer er hoge eisen worden gesteld aan de performance van het systeem, of wanneer fouten in de specificatiefase ernstige gevolgen zouden hebben (denk bijv. aan programmatuur voor medische- of defensiedoeleinden).
Van het grootste belang in een gegeven situatie is een goede keuze van de juiste methode, d.w.z. een voor het betreffende (informatie-)probleem, de omgeving en het type organisatie geschikte methode.
Globaal gesproken zijn er twee benaderingswijzen te onderscheiden bij ontwikkelingsmethodieken voor informatiesystemen, te weten de procesbenadering en de ge-

gevensbenadering. Beide benaderingen hebben zowel voor- als nadelen. De benaderingsvoorkeur is geen kwestie van smaak, maar dient een weloverwogen keus te zijn. Het is echter ook duidelijk dat processen en gegevens, als systeemcomponenten, beide aandacht dienen te krijgen.

In bepaalde probleemsituaties geniet een procesbenadering duidelijk de voorkeur. Hierbij kiest men de uit te voeren functies als startpunt en vervolgens inventariseert en structureert men de daarvoor benodigde gegevens. De betekenis van de gegevens wordt hier ontleend aan de processen die ermee moeten worden uitgevoerd. Voor veel gebruikers van toekomstige informatiesystemen zijn de functies van die processen concreter dan de gegevens, omdat de dynamische dagelijkse praktijk meer uit deze functies dan uit de gegevens spreekt.

In andere situaties – en zeker wanneer de bestaande activiteiten niet zo duidelijk te beschrijven zijn – is een gegevensbenadering op zijn plaats. Hierbij worden de gegevens onafhankelijk van de processen, die ermee moeten werken, beschreven en wordt de gegevensstructuur bepaald door de eigenschappen van de gegevens en niet door het gebruik dat van de gegevens wordt gemaakt. Men neemt hier als het ware afstand van de problemen van gebruikers en benadert deze problemen op een vrij hoog abstractieniveau. Het feit dat men bij een gegevensbenadering gebruikers dwingt te abstraheren van hun concrete praktijksituatie is een groot nadeel. Veel gebruikers van het toekomstige informatiesysteem kunnen dat blijkbaar niet (zie ook [Bemel83b]). Gegevens zeggen hun alleen iets in relatie met de (verwerkings)processen. In hun perspectief zijn de gegevens een afgeleide. De procesbenadering doet dat ook. Dat is een groot voordeel van de procesbenadering, waarbij wel opgemerkt dient te worden, dat de gegevenstypen, die op deze wijze ontstaan, in principe een beperkte 'reikwijdte' hebben, d.w.z. ze kunnen nogal lokaal gekoppeld zijn aan de processen.

Wanneer men de momenteel bestaande op een procesbenadering gebaseerde methoden en technieken vergelijkt vallen een aantal overeenkomsten op. Om de complexiteit te beheersen hanteren deze methodieken vrijwel allemaal een top-down aanpak, waarbij de in eerste instantie globale concepten in een aantal stappen nader gedetailleerd worden. De algemene tendens uit de serie ontwikkelingsmethoden in dit tijdschrift ([Bemel83a], [NGI83]) is verder dat grafische beschrijvingswijzen op dit moment de voorkeur hebben boven tekstuele, omdat de daarbij gehanteerde centraal staande schematechnieken zeer geschikt zijn als communicatiemiddel tussen informatie-analisten/systeemontwerpers en eindgebruikers, en later gebruikt kunnen worden als documentatiemiddel. Methodieken voor het op deze wijze analyseren van probleemsituaties en het ontwerpen van informatiesystemen zijn echter erg arbeidsintensief, zeker wanneer systeemspecificaties nog voortdurend kunnen wijzigen, hetgeen in de (her)ontwerpfase altijd het geval is. Dergelijke methodieken hebben dan ook een gemeenschappelijk probleem: het onderhouden, en bewaken van de consistentie en volledigheid, van de grote en explosief groeiende hoeveelheden informatie, ontstaan in het top-down analyse- en ontwerpproces, blijkt een onevenredig groot deel van de tijd van de informatie-analist/systeemontwerper op te eisen, en in bepaalde gevallen zelfs ondoenlijk te zijn. De op zich goede methode kan en zal dus niet op de juiste wijze gehanteerd worden, hetgeen onvermijdelijk leidt tot een lagere productiviteit van de informatie-analist, een slechtere kwaliteit van het functioneel ontwerp, en uiteindelijk: afschaffing van de methode.

Een geautomatiseerd hulpmiddel zou voor de geconstateerde problemen een oplossing kunnen bieden en het juiste gebruik van de methodiek beter kunnen afdwingen. Om de hier eerder genoemde redenen ligt het voor de hand om van dit hulpmiddel tevens te eisen dat een grafische beschrijvingswijze voor de gebruikers voorhanden is. Bovendien zou het aanbevelenswaardig zijn indien het gerealiseerd werd als een portable hulpmiddel, zodat de informatie-analist dit gereedschap met weinig moeite mee zou kunnen nemen naar de gebruiker. Reeds bestaande hulpmiddelen – zoals bijvoorbeeld PSL/PSA, dat reeds sinds 1968 grote bekendheid geniet en als een belangrijke exponent van de vierde generatie systeemontwikkelingsmethodieken kan worden beschouwd – voldoen niet aan dergelijke eisen [Gerst83].

In dit artikel wordt een onderzoek beschreven dat betrekking heeft op het ontwerpen en op een micro-computer gedeeltelijk realiseren van een dergelijk hulpmiddel voor de door Desisco Nederland B.V. (een adviesbureau voor informatiesystemen) ontwikkelde MOS methodiek [Wagen81]. Hierbij is een prototyping-aanpak gevolgd, met name omdat er sprake bleek te zijn van een onduidelijke en instabiele informatiebehoefte. De eerste delen van dit prototype zijn momenteel operationeel. Het systeem, dat gebouwd wordt op een VICTOR micro-computer, moet eind 1985 geheel gerealiseerd zijn.

Achtereenvolgens zullen we in een aantal hoofdstukken verslag doen van dit onderzoek, waarbij we de volgende indeling zullen hanteren. In hoofdstuk 2 zal kort worden ingegaan op de rol die functionele specificaties spelen in de informatiesysteem-levenscyclus, waarna enkele redenen zullen worden genoemd die ertoe leiden dat de specificatie- en ontwerpfasen niet de aandacht verkrijgen die ze verdienen. Vervolgens wordt in hoofdstuk 3 gekeken naar de eisen die men mag stellen aan een geautomatiseerde ondersteuning, en in hoofdstuk 4 naar bestaande geautomatiseerde hulpmiddelen. Hoofdstuk 5 is gewijd aan de in het voornoemde onderzoek ontwikkelde tool en de tot dusverre behaalde resultaten. Tenslotte zullen we in hoofdstuk 6 een aantal conclusies trekken. Een nadere uitwerking van voorafgaande onderwerpen kan men desgewenst vinden in [Vonk84a].

Dit artikel is vooral van belang voor diegenen die verantwoordelijk zijn voor de keuze en het beheer van methoden en technieken voor het ontwikkelen van informatiesystemen, voor diegenen die zich bezighouden met de ontwikkeling van geautomatiseerde gereedschappen die voornoemde methoden ondersteunen, en voor informatie-analisten, die gebruik zullen gaan maken van gereedschappen als hier beschreven. De auteurs hopen een discussie op gang te brengen over het gepresenteerde architectuurconcept en de uitwerking daarvan.

Ten aanzien van de gehanteerde terminologie nog het volgende: de lezer(es) wordt verzocht om waar dit van toepassing is in plaats van 'hij' resp. 'zijn', 'hij/zij' resp. 'zijn/haar' te lezen. Aanduidingen als 'informatie-analist', 'systeem-ontwerper', 'gebruiker' etc. impliceren géén aanname omtrent de sexe van de desbetreffende persoon.

2 HET DOEL VAN SPECIFICATIES

Functionele specificaties vormen een precieze beschrijving van de eisen die er aan het te ontwikkelen systeem worden gesteld. Zij impliceren als het ware een begrenzing van de verzameling van mogelijke oplossingen, en vormen de basis voor het systeemontwerp (het systeemontwerp is de uiteindelijk uit die verzameling gekozen oplossing). In de testfase kan met behulp van het functioneel ontwerp worden nagegaan of het gerealiseerde systeem inderdaad voldoet aan de gestelde eisen. Specificaties maken validatie en verificatie mogelijk. Voorts spelen specificaties een rol bij het aanpassen of uitbreiden van een informatiesysteem. Met behulp van de functionele specificaties en het systeemontwerp kunnen de te wijzigen delen snel gelokaliseerd worden, hetgeen met alleen de source-code – zeker bij slecht gestructureerde programma's – welhaast onmogelijk zou zijn. Tenslotte vormen functionele specificaties de basis voor de communicatie tussen en binnen de verschillende bij het ontwikkelingsproces betrokken groepen belanghebbenden (informatie-analisten/systeemontwerpers, projectleiders, managers en gebruikers)

De tot de diverse methodieken behorende schematechnieken spelen in de specificatie-fasen een belangrijke rol. Zij ondersteunen de informatie-analist in het *analyse-* en *ontwerp*proces. Wanneer de toekomstige gebruiker van het te ontwikkelen informatiesysteem bereid is om de notatiewijze te leren begrijpen kunnen de schema's een gemeenschappelijke taal gaan vormen, en als zodanig een belangrijke stimulans betekenen voor de *communicatie* tussen de ontwikkelaar en de eindgebruiker. Gebruikmaking van deze schematechnieken heeft nog een derde positief gevolg: de (systeemontwikkelings)*documentatie* zal altijd up-to-date zijn.
In een aantal studies is aangetoond dat het in de bouwfase herstellen van door een onvolledig, onjuist of vaag functioneel ontwerp ontstane fouten zeer veel duurder is dan het herstellen van diezelfde fouten in de specificatiefase. In [Strau83] wordt naar aanleiding van een IBM-studie gesteld dat het hier een faktor 80 betreft; in [Teich80] wordt zelfs gesproken over een faktor 400. Volgens [Strau83] vindt 60% van de in de bouwfase te herstellen fouten zijn oorsprong in de specificatie- en ontwerpfasen, en heeft 80% van de 'onderhouds'-inspanning betrekking op het herstellen van deze fouten. Desondanks krijgt de specificatiefase vaak nog steeds niet de aandacht die ze verdient. Dit heeft een aantal oorzaken.
Ten eerste hecht men vaak overdreven grote waarde aan de snelheid waarmee een systeem wordt gerealiseerd [Baber82]. Functionele specificaties worden als onnodige, vertragende ballast beschouwd. De ontwikkelaars van het systeem hebben soms zelf geen behoefte aan volledig uitgewerkte specificaties, en moeten deze vervaardigen ten behoeve van het onderhoud van het systeem. Het documenteren van alle eisen en aannames wordt vaak beschouwd als zijnde het vervelendste onderdeel van het gehele ontwikkelingsproces.
Ten tweede is het doorgaans bijzonder eenvoudig en aantrekkelijk om, in die gevallen waarin door gebruikers – minieme – wijzigingen worden verlangd, alléén de programmatuur aan te passen, en de specificatie- en ontwerpgegevens ongewijzigd te laten. Vanaf dat moment zijn specificaties niet meer betrouwbaar, en dus niet meer geschikt als basis voor onderhoud.
Ten derde zijn specificaties in vele gevallen niet stabiel [Davis82]. De informatiebehoeften van gebruikers kunnen sterk wijzigen na invoering van een systeem, of zelfs al nadat de specificaties zijn doorgesproken en de gebruikers een duidelijker beeld hebben verkregen van het systeem. Informatie-analisten verzuchten in een dergelijk geval dat gebruikers niet weten wat ze willen, maar zien daarbij over het hoofd dat het juist één van de doelen van de specificatiefase is om de communicatie tussen gebruikers en analisten te stimuleren zodat er een goed beeld van het probleem verkregen kan worden. Het handmatig opstellen en telkens weer wijzigen van diagrammen, teksten, tabellen, lijsten etc, vergt echter zeer veel tijd. Ook om die reden zullen informatie-analisten vaak er toe neigen het documenteren zo lang mogelijk uit te stellen.
Een vierde reden voor de geringe aandacht die de specificatiefase ontvangt is gelegen in de inspanning die het consistent houden van specificaties met zich meebrengt. Tussen verschillende documenten bestaan vaak allerhande, soms ondoorzichtige, verbanden. Gegevens zijn vaak redundant aanwezig; elke mutatie brengt een zeker gevaar van inconsistentie met zich mee. Schema's zijn onderling sterk aan elkaar gekoppeld, en de in één schema aangebrachte wijzigingen kunnen allerhande op het eerste gezicht onverwachte gevolgen hebben in een groot aantal andere schema's, zowel op hetzelfde als op andere abstractieniveaus. Het achterhalen van deze gevolgen is bijzonder tijdrovend.
Voor de gesignaleerde problemen zijn een aantal remedies denkbaar. Zo kan men de ontwikkelaar geen andere keus laten dan bij elke wijziging ook de specificaties aan te passen. Dit kan bijvoorbeeld bereikt worden door stringente wijzigingsprocedures te introduceren. Een betere oplossing is echter het opstellen van, en aanbrengen van wijzigingen in, specificaties te vereenvoudigen door voor de functionele ontwerpfase geautomatiseerde hulpmiddelen te ontwikkelen. Men kan voor dergelijke gereedschappen dan ook een levendige belangstelling constateren.

3 EISEN TEN AANZIEN VAN GEAUTOMATISEERDE HULPMIDDELEN

Een, het ontwerpproces ondersteunend, gereedschap moet de communicatie tussen de diverse bij het ontwikkelingsproces betrokken belangengroepen vereenvoudigen [Ridd80]. Elke gebruiker van het gereedschap moet inzicht kunnen verkrijgen in het systeem in wording. Tools moeten beschrijvingen die verschillende aspecten van het te ontwikkelen systeem belichten kunnen produceren, ook wanneer deze informatie niet expliciet is ingevoerd. Er dient daarbij sprake te zijn van een grote flexibiliteit ten aanzien van rapporteringsmogelijkheden. Het gereedschap moet gekoppeld zijn aan een bepaalde ontwikkelingsmethodiek [Ridd80]. Tools moeten het gebruik van die methodiek bevorderen en ondersteunen, en de effecten van toepassing van deze methodiek zo mogelijk versterken. De binnen de desbetreffende methodiek gehanteerde analyse- en ontwerpdocumenten moeten direkt als invoer geaccepteerd worden: er mag geen sprake zijn van een 'vertaalslag' tussen werkdocument en voor de tool acceptabele invoer. De verzameling hulpmiddelen moet het gehele informatiesysteemont-

wikkelingstraject ondersteunen ([Wass82a], [Wass82b]). Met de introductie van één groot, het ontwikkelen ondersteunend, systeem loopt men het gevaar van inflexibiliteit. Door de introductie van een verzameling gereedschappen bestaat er een kans dat deze niet op elkaar afgestemd zijn. Idealiter dient er daarom sprake te zijn van een geïntegreerde verzameling gereedschappen. Elk onderdeel van dit, het ontwikkelen van informatiesystemen ondersteunend, systeem dient één bepaalde taak, of een aantal zeer sterk aan elkaar gerelateerde taken, te verrichten. De verschillende gereedschappen moeten een voor de gebruiker uniform uiterlijk bezitten. Tools moeten het de gebruiker mogelijk maken om met verschillende versies van het systeem in wording te werken. De ontwikkelaar moet zich op een eenvoudige wijze door de 'versie-ruimte' kunnen bewegen. Verder moet het gereedschap informatie met betrekking tot het gebruik ervan bijhouden. Op deze wijze kan het projectmanagement meer inzicht in de effectiviteit van de tool, en de in bepaalde fasen van het ontwikkelingstraject benodigde inspanning, verkrijgen.
Het blijkt bijzonder moeilijk te zijn om aan alle voorgaande eisen te voldoen. Enkele redenen daarvoor zijn dat er nog vrijwel geen methodieken bestaan, die het gehele ontwikkelingsproces van begin tot eind ondersteunen, en dat het vrij kostbaar is om tools voor eerdere ontwikkelingsfasen te produceren (met slechts een geringe kans op succes) [Ridd80]. Een 'evolutionaire' ontwikkelingsaanpak (begin met een 'minimale' tool, en ontwikkel verder n.a.v. ervaringen en reacties) biedt het beste perspectief op een uiteindelijk succesvolle ontwikkeling [Nassi80].

4 INVENTARISATIE VAN GEAUTOMATISEERDE HULPMIDDELEN

Er bestaat momenteel reeds een groot aantal geautomatiseerde gereedschappen ten behoeve van het informatiesysteemontwikkelingsproces. De huidige situatie wordt gekenmerkt door een zeer groot aantal gereedschappen voor de bouwfase, en een minieme hoeveelheid hulpmiddelen voor de specificatie-, ontwerp- en testfase. Dit is niet verwonderlijk: gereedschap wordt doorgaans ontwikkeld als reactie op een waargenomen probleem [Hecht82]. Men wil iets mogelijk maken dat voorheen economisch of fysiek gezien niet haalbaar was. Zeker de in de industrie ontwikkelde, het informatiesysteemontwikkelingsproces ondersteunende, gereedschappen getuigen vaak van een zeer pragmatische aanpak, en zijn primair gericht op het ondersteunen van dat deel van het ontwikkelingsproces dat als buitengewoon problematisch ervaren wordt [Haus82]. De problemen lagen in het verleden met name in de *bouw*fase. De programmeur kan heden ten dage dan ook beschikken over compilers/interpreters, linkers, debuggers, editors, etc. Vrij nieuwe hulpmiddelen voor deze fase zijn de programma- en applicatiegeneratoren. Er is tot dusverre echter nauwelijks sprake geweest van een samenhangende verzameling gereedschappen. Hier wordt wel aan gewerkt: zo heeft het U.S. Department of Defense APSE gespecificeerd, een geïntegreerde verzameling tools ten behoeve van de ontwikkeling van ADA-programmatuur. Een ander voorbeeld van een geïntegreerde 'Programming Environment' is PWB/UNIX (van Bell Telephone Laboratories) dat de programmeur een consistente, samenhangende verzameling krachtige faciliteiten biedt ([Dieb82], [Haus82], [Hend82]).
Het aantal gereedschappen dat in de overige fasen toegepast kan worden is echter groeiende. Een tool die sterk de nadruk legt op de *test*-fase is HDM [Haus82]. Tot de tot dusverre voor de *specificatie-* of *ontwerp*fase gecreëerde gereedschappen behoren o.a. PSL/PSA (University of Michigan, [Teich77], [Gerst81]), SREM (van TRW, [Bell77], [Smith80]), AUTOIDEF (van Boeing, [Smith80]), IAST (Control Data, [CDC82]), SPECIF (IGL, [Lissa84]) en de SA Support Tools (ontwikkeld door Tektronix, [Delis82]). Een aantal organisaties/instanties tracht een geïntegreerde set hulpmiddelen die het gehele ontwikkelingstraject dekt te ontwikkelen. Een ambitieus voorbeeld van zo'n verzameling is SDS (Software Development System), ontwikkeld ten behoeve van zogenaamde Ballistic Missile Defense (BMD) Systems [Davis77]. Overigens is het reeds genoemde SREM een onderdeel van dit systeem. Het is echter nog niet duidelijk of en hoe de diverse onderdelen van SDS op elkaar aansluiten [Haus82]. ICAS (Hitachi, [Koba83]) en PROMOD (GEI, [Hrus83]) zijn andere voorbeelden van geïntegreerde het gehele ontwikkelingstraject ondersteunende (gedeeltelijk reeds gerealiseerde) verzamelingen gereedschappen.
Veel van de genoemde ondersteuningen behoren bij een bepaalde methodiek. Zo is AUTOIDEF gecreëerd als ondersteuning voor IDEF, SPECIF voor SADT en IAST voor NIAM, behoren de SA Support Tools bij Structured Analysis van DeMarco, en is de 'COGITOR Systeemontwikkelingsmachine' (CAP GEMINI, [Schel81]) ontwikkeld ten behoeve van Systematrix. Deze bij een methodiek behorende gereedschappen zijn vrijwel alle van een vrij recente datum. Andere gereedschappen – zoals programmageneratoren – zijn doorgaans methode-onafhankelijk (kunnen los van een methode gebruikt worden), waarmee overigens niet gezegd is dat deze niet in een bepaalde methode ingepast zouden kunnen worden. Naast de bij één methode behorende, en de in principe methode-onafhankelijke gereedschappen, is er ook een categorie tools die geschikt is voor verschillende methodieken. Een hulpmiddel als PSL/PSA kan (met behulp van een meta-systeem, [Gerst81]) geschikt worden gemaakt voor de eigen methoden. ARGUS (een door Boeing gerealiseerde ondersteuning) zal zowel Structured Design als SADT (van SofTech) en SAMM (van Boeing) ondersteunen [Haus82].
Beziet men de momenteel bestaande gereedschappen voor de specificatie- en ontwerpfasen, dan vallen al snel een aantal overeenkomsten op. Deze hulpmiddelen zijn vrijwel allemaal opgebouwd rond een 'design-database', waarin alle uit het analyse- en ontwerpproces voortvloeiende informatie wordt opgeslagen ([Hend82], [Haus82]). Op deze wijze kan redundantie beheerst worden, en de integriteit (consistentie, volledigheid) gehandhaafd worden. Teneinde de analyse- en ontwerpresultaten te kunnen opslaan – de database te vullen – moeten deze worden weergegeven in een formele taal. Voorbeelden van een dergelijke beschrijvingstaal zijn RSL (bij SREM) en PSL. Naast dergelijke één-dimensionale talen ziet men steeds meer de mogelijkheid om – op een interactieve wijze – de bij een bepaalde methodiek behorende diagrammen op te bouwen, te onderhouden en op te slaan (bijv. SREM, AUTOIDEF, SA Support Tools, ARGUS). Bij PSL/PSA ontbreekt een dergelijke facili-

teit nog [Gerst81]. Het werkdocument van de informatie-analist/systeemontwerper (zoals informatieprecedentieschema's en Nassi-Shneidermandiagrammen) kan noch als invoermedium gebruikt worden, noch als uitvoer geproduceerd worden door de beschikbare PSL/PSA-versie [Gerst83]. Er bestaat overigens weinig twijfel over dat twee-dimensionale (grafische) talen superieur zijn aan tekstuele beschrijvingswijzen, zowel wat betreft de snelheid van informatieoverdracht als qua helderheid en duidelijkheid. Daartegenover staat dat sommige zaken nauwelijks grafisch weergegeven kunnen worden (denk bijv. aan bepaalde constraints).
Nadat de analyse- en ontwerpresultaten zijn weergegeven in de bij een bepaalde geautomatiseerde ondersteuning behorende formele beschrijvingstaal zorgt een onderdeel van dit systeem voor een syntax-check en opslag in de database. Naast de component die ervoor zorgdraagt dat de database gevuld raakt bestaat er doorgaans een component waarmee de verzameling opgeslagen ontwerpgegevens geanalyseerd kan worden. In de meeste gevallen kunnen hiermee rapporten gegenereerd worden waarin handmatig kan worden nagegaan of de database geen tegenstrijdigheden bevat of incompleet is; soms kunnen de aan het systeem bekende constraints echter automatisch bekrachtigd worden.
Bepaalde tools kunnen afleidbare specificaties (specificaties die impliciet in de database aanwezig zijn, doch die niet expliciet zijn ingebracht) produceren. Deze gereedschappen nemen dus bepaalde (formaliseerbare) taken over van de informatie-analist of systeemontwerper. Daarnaast verschaffen dergelijke hulpmiddelen een zekere flexibiliteit: de opgeslagen analyse-/ontwerpinformatie kan naar verschillende gezichtpunten (aangepast aan de individuele voorkeur op een bepaald moment) worden weergegeven. Veel gereedschappen hebben een component waarmee allerhande overzichten van de inhoud van de database kunnen worden geproduceerd. De opgeslagen ontwerpinformatie kan op verschillende wijzen doorlopen worden; soms met vooraf gedefinieerde queries, soms met een on-line query-taal. Doorgaans kunnen ook gecomprimeerde overzichten gegenereerd worden. Meestal is alle uitvoer zowel op een (grafische) terminal als op een (grafische) printer of plotter te verkrijgen.
De gereedschappen zijn veelal op een mainframe gerealiseerd [Smith80]. Zowel de kosten van gebruik als de responsetijden (time-sharing omgeving, veel operaties op een database) zijn hoog. Omdat deze gereedschappen vaak gebruik maken van DVST-beeldschermen (Direct-View Storage Tube oftewel geheugenbuis; het gehele scherm moet telkens ververst worden) wordt de gebruiker van bijvoorbeeld SREM of AUTOIDEF niet met de meest optimale interface geconfronteerd: over drukke communicatielijnen kan het erg lang duren voordat een compleet diagram is opgebouwd [Smith80].
Er zijn verschillende redenen waarom tools doorgaans in een conventionele mainframe-omgeving worden gerealiseerd. In de grote organisaties die geautomatiseerde gereedschappen kunnen en willen realiseren zijn doorgaans al mainframes met bijbehorende ontwikkel-tools (editors, compilers, DBMS'n) aanwezig. De ontwikkelaars van geautomatiseerde hulpmiddelen werken doorgaans al met deze apparatuur en programmatuur, en door het systeem op een mainframe te realiseren kunnen de investeringen in eerste instantie bescheiden blijven. De steeds krachtiger en goedkoper wordende mini's en – vaak al over uitgebreide grafische mogelijkheden beschikkende – micro's, en de voortschrijdende databasetechnologie (gedistribueerde databases, krachtige pakketten op micro's), bieden de gereedschapsmaker echter nieuwe mogelijkheden.

5 EEN REALISATIE

In dit hoofdstuk wordt een voor de MOS methodiek in ontwikkeling zijnde geautomatiseerde ondersteuning beschreven. Allereerst wordt in par. 5.1 aangegeven welke werkwijze gevolgd wordt bij deze ontwikkeling. Par. 5.2 bevat een functionele beschrijving van de subsystemen van de tool. Tenslotte wordt in par. 5.3 ingegaan op de huidige status van het project, en op enkele mogelijke toekomstige ontwikkelingen.

5.1 Een prototyping benadering

In de inleiding is reeds gesteld dat de keuze van ontwikkelingsaanpak grotendeels moet afhangen van karakteristieken van de probleemsituatie en van het te ontwikkelen informatiesysteem. Hierbij komen vragen aan de orde als:

- is het mogelijk om tot *stabiele* specificaties te komen? (Weten gebruikers wat ze nu en in de toekomst willen? Kunnen ze hun wensen c.q. eisen duidelijk verwoorden, spelen subjectieve eisen een grote rol, heeft de informatie-analist ervaring met dergelijke systemen?)
- zijn er duidelijke *processen* te onderscheiden?
- wat is de *omvang*/de *complexiteit* van het te ontwikkelen informatie-systeem?

Een factor die bij de keuze van de wijze waarop het MOS-ondersteunende meta-informatiesysteem ontwikkeld zou moeten gaan worden een belangrijke rol speelde was de *onduidelijke informatiebehoefte*. In diverse informele gesprekken met informatie-analisten bleek dat deze weliswaar de noodzaak van een dergelijke ondersteuning erkenden, maar dat niemand precies kon omschrijven welke functies een dergelijk systeem zou moeten omvatten. Men had slechts een vaag idee omtrent de mogelijkheden die een geautomatiseerde ondersteuning zou kunnen bieden. Er mocht dan ook verwacht worden dat er na een eerste realisatie een sterke evolutie in de informatiebehoeften zou plaatsvinden. Het doorlopen van het gehele systeemontwikkelingstraject zou hoogstwaarschijnlijk tot gevolg hebben dat de eerste ('tastbare') resultaten later dan voorzien c.q. wenselijk beschikbaar zouden komen, het project duurder zou worden dan toelaatbaar geacht werd, en het uiteindelijke product niet lang of in het geheel niet aan de gebruikerswensen zou kunnen voldoen. Een dergelijke aanpak zou de kans op een succesvolle afronding van het project aanzienlijk verkleinen. Om deze reden is gekozen voor een andere benadering, met een meer iteratief karakter: prototyping ([Beek82], [Vonk84b]).
In het navolgende zal worden toegelicht wat in het onderzoek nu precies onder een prototyping-aanpak verstaan werd, en welke fasen men daarbij kan onderscheiden.

1. *Kort vooronderzoek.* In dit vooronderzoek wordt getracht om een globaal beeld van de informatiebehoefte en van de wensen/eisen ten aanzien van het te ontwikkelen informatiesysteem te verkrijgen.

Dergelijke gegevens kunnen worden achterhaald door het houden van (semi-gestructureerde) interviews. Indien het een onbekend gebied betreft voor zowel gebruiker als informatie-analist kan het bijzonder zinvol zijn om de beschikbare literatuur te bestuderen.

2. *Globale functie- en gegevens-analyse.* Hoewel een van de doelen van de prototyping-aanpak het verkrijgen van een gedetailleerd inzicht in de gewenste functies van het informatiesysteem is zal er toch een zekere basis voor de prototype-bouw dienen te bestaan. Met behulp van een globale functie-analyse wordt daarom getracht om een indruk te krijgen van de belangrijkste functies van het systeem. Indien het een omvangrijk systeem betreft kan men reeds een opdeling in separaat te ontwikkelen subsystemen trachten te verkrijgen. Ook de gegevens-analyse heeft in eerste instantie een globaal karakter. Het resultaat van de gegevens-analyse (een gegevensmodel) kan echter reeds een stabiele basis vormen voor de programmeerfase, zeker wanneer men de beschikking heeft over een goed DBMS.
3. *Programmering.* Ondersteund door goede prototyping-hulpmiddelen, en nog met weinig aandacht voor de efficiency van het uiteindelijke product, wordt een eerste versie van het informatiesysteem gebouwd. Het betreft hier een operationeel prototype; men beperkt zich dus niet tot het simuleren van het uiteindelijke systeem. Deze fase wordt afgesloten met een werkend prototype, systeemdocumentatie, en nog zeer beperkte gebruikersdocumentatie.
4. *Evaluatie.* Het ontwikkelde prototype wordt, liefst in een werkelijke praktijksituatie (pilot-project), geëvalueerd door toekomstige gebruikers.

Wanneer in de laatste fase blijkt dat het systeem aanpassingen behoeft, worden de fasen 2, 3 en 4 wederom doorlopen. De gegevens- en functiemodellen worden aangepast, uitgebreid of gedetailleerd, de programmatuur wordt aangepast, en vervolgens vindt er een nieuwe evaluatie plaats. Dit proces wordt herhaald totdat het prototype – grotendeels – voldoet aan de gebruikerswensen/-eisen. Er is dus sprake van een iteratieve aanpak. In eerste instantie bestaat daarbij veel aandacht voor het pragmatische en semantische aspect (WAT moet er komen); in een latere fase komen ook syntactische en technische aspecten aan de orde (HOE realiseren). Dan gaan ook de in eerdere evaluaties genoteerde prestatie-eisen een rol spelen.

Op het moment dat een op de in het voorafgaande beschreven wijze gerealiseerd prototype grotendeels voldoet aan de gebruikerswensen en -eisen bestaan er een tweetal mogelijkheden. Men kan – nu op basis van goede functionele specificaties – het systeem op een gestructureerde wijze opnieuw bouwen, of men kan het informatiesysteem als 'af' beschouwen. In de meeste gevallen (afhankelijk van de gehanteerde prototyping hulpmiddelen) zal de eerste benadering de voorkeur verdienen, met name omdat het moeilijk zal zijn om de ontstane programmatuur te onderhouden. Het is echter niet altijd haalbaar om het systeem opnieuw te bouwen. In beide gevallen zal men nog aandacht dienen te besteden aan het verzorgen van gebruikersdocumentatie, evt. opleidingen, en aan aspecten die betrekking hebben op de invoering van het ontwikkelde systeem.

5.2 **Functionele beschrijving meta-informatiesysteem**

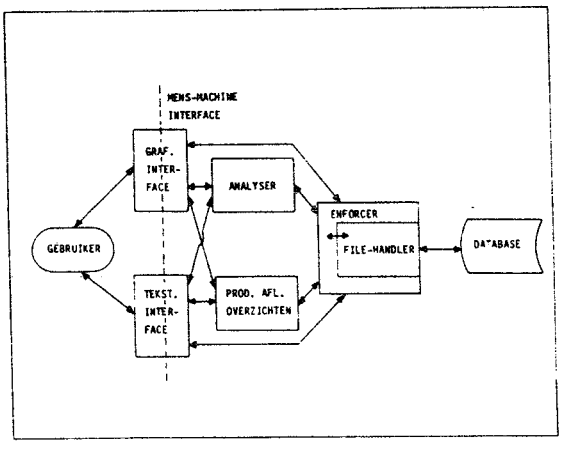
Voordat overgegaan kon worden tot de bouw van een prototype van het informatiesysteem moest – zoals gesteld – globaal worden bepaald welke taken het systeem uiteindelijk zou moeten verrichten. Na een kort vooronderzoek bleek dat het gereedschap uiteindelijk de in het navolgende opgesomde functies zal moeten bezitten:

- in de eerste plaats zal de 'tool' de informatie-analist/systeemontwerper moeten ondersteunen bij het opbouwen, opslaan en onderhouden van de in het analyse- en ontwerpproces ontstane tekstuele en grafische informatie. Het systeem moet zowel het op interactieve wijze opbouwen van deze schema's (met behulp van een grafische invoerfaciliteit, bijv. een lichtpen) als de productie van hard-copy's op bijv. een grafische printer ondersteunen;
- een tweede belangrijke taak van dit hulpmiddel is het bewaken van de integriteit van de 'design-database'. De informatie-analist moet door het systeem gewezen worden op onduidelijkheden, onvolledigheden en inconsistenties. Inconsistente toestanden van de database mogen niet geaccepteerd worden;
- ten derde moet het systeem diverse routinematige werkzaamheden van de informatie-analist overnemen. Schema's en overzichten die uit de opgeslagen ontwerpinformatie zijn af te leiden zullen automatisch gegenereerd moeten kunnen worden. Afleidbare wijzigingen moeten automatisch doorgevoerd worden.

Het systeem zou gerealiseerd gaan worden op een microcomputer, zodat de informatie-analist dit gereedschap met weinig moeite mee zou kunnen nemen naar een opdrachtgever.

Het was duidelijk dat de realisatie van een dergelijk omvangrijk meta-informatiesysteem een complexe zaak zou worden. Om het ontwikkelingsproces beter beheersbaar te maken werd het systeem opgedeeld in een aantal subsystemen. Deze opdeling was gebaseerd op een aantal overwegingen. Zo stelt in het algemeen het verwerken van grafische informatie duidelijk andere eisen dan het opslaan van tekstuele informatie. Daarnaast is het aanbevelenswaardig om constraints door één subsysteem te laten bekrachtigen. (Onbeheerste redundantie van constraints is even schadelijk als onbeheerste redundante

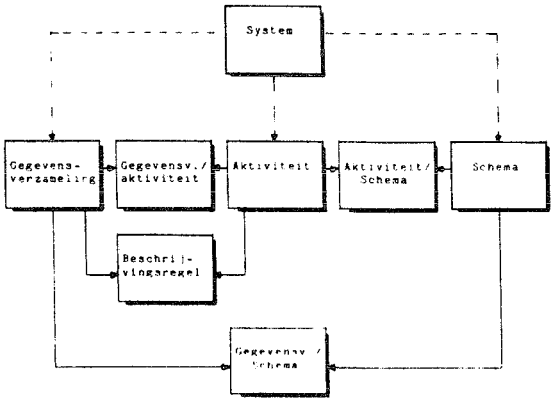
Figuur 1: Structuur meta-informatiesysteem



opslag van gegevens!) Voorts dient men altijd te streven naar modules van een zodanige omvang dat deze hanteerbaar blijven, en tenslotte zal men er voor moeten zorgdragen dat subsystemen zo onafhankelijk mogelijk zijn en onafhankelijk van elkaar ontwikkeld kunnen worden. In figuur 1 wordt de opdeling weergegeven.
In het navolgende zullen deze subsystemen achtereenvolgens beschreven worden.

5.2.1 *De File-Handler*
Het centrale gedeelte van het onderhavige meta-informatiesysteem wordt gevormd door de verzameling opgeslagen ontwerpgegevens. Aan de *structuur* van deze verzameling wordt aandacht geschonken in een gegevensmodel, waarin wordt weergegeven in welke objecttypen in het objectsysteem (het in beschouwing genomen gedeelte van de wereld, bekeken als systeem) we geïnteresseerd zijn, en welke structurele afhankelijkheden er tussen die objecttypen bestaan. De gegevensstructuur moet – teneinde inconsistenties ten gevolge van updates zoveel mogelijk te voorkomen – genormaliseerd zijn. In een functioneel ontwerp zijn doorgaans dezelfde ontwerpgegevens op diverse plaatsen, in verschillende verschijningsvormen, aanwezig, hetgeen het (onopzettelijk) introduceren van tegenstrijdigheden natuurlijk in de hand werkt.
De *File-Handler* verzorgt het onderhoud van de op een netwerkachtige wijze gestructureerde database. Het netwerkschema in figuur 2 bevat meer informatie over de structuur van deze database. De gehanteerde schematechniek komt grotendeels overeen met de welbekende Bachman-notatie: rechthoeken (met uitzondering van SYSTEM) geven recordtypen weer, pijlen (die van owner naar member wijzen) geven 1:n relaties weer. De structuur van deze database is gebaseerd op de resultaten van de eerder in het onderhavige onderzoek uitgevoerde gegevens-analyse; de (genormaliseerde) logische structuur is afgebeeld in het DBTG-model.
De File-Handler voert opdrachten uit om ontwerpgegevens op te slaan, te wijzigen of op te zoeken in, c.q. te verwijderen of op te halen uit, de database. Deze opdrachten zijn afkomstig van andere subsystemen en worden via de DML, die hier niet nader beschreven zal worden, overgebracht. Het behoort niet tot de taak van de File-Handler om na te gaan of dergelijke opdrachten inconsistentie of onvolledigheid van de database tot gevolg zouden kunnen hebben: bekrachtiging van de diverse (statische en dynamische) constraints wordt verzorgd door de in de volgende paragraaf te behandelen Analyser en Enforcer.

Figuur 2: Structuur 'design-database'

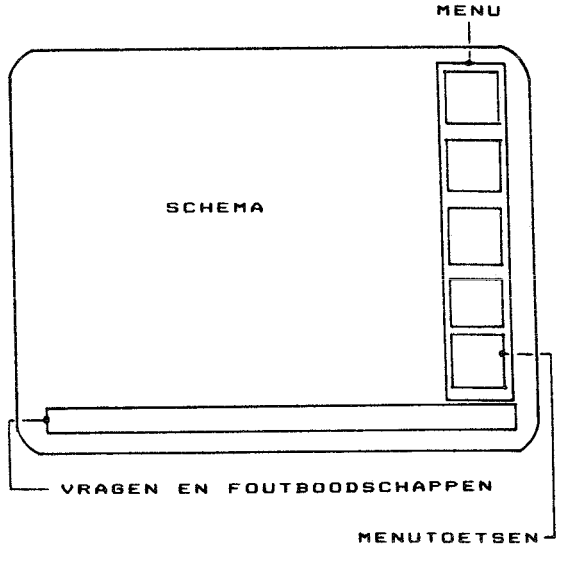


5.2.2 *De Analyser en Enforcer*
Een beschrijving van het objectsysteem is niet compleet zonder een beschrijving van de *regels* oftewel *constraints* die voor de in beschouwing genomen objecttypen gelden (ISO '100% principle', [Griet82]). Dergelijke regels bepalen de op elk moment toegestane toestanden, en overgangen tussen toestanden, van de inhoud van de database. Voor een beschrijving van de structuur van een gegevensverzameling die gecompleteerd is met de voor die gegevensverzameling geldende regels wordt momenteel de term 'conceptueel schema' gehanteerd. Meerdere van de moderne informatieanalyse-methoden als NIAM en INFOMOD [Griet84] besteden veel aandacht aan het achterhalen en weergeven van constraints, terwijl dit aspect in reeds langer bestaande methoden zoals Bubble Charting vaak grotendeels buiten beschouwing blijft.
Door de regels waaraan het informatiesysteemontwerpproces onderworpen is te expliciteren en vervolgens aan het meta-informatiesysteem kenbaar te maken, kan dit vaststellen of ingevoerde informatie accuraat is. (Een bijkomend voordeel daarvan is dat deze regels nu vastliggen en niet meer verloren kunnen gaan. De methode wordt 'beter te leren'.) De gebruiker kan ervan op aan dat de database te allen tijde zowel compleet als consistent (geen tegenstrijdigheden bevattende) is: de *integriteit* van de database is gewaarborgd. Dit is zeer essentieel! Een gebruiker die niet kan vertrouwen op de juistheid en volledigheid van de door dit informatiesysteem verstrekte gegevens zal er al spoedig geen gebruik meer van maken. Voorts zal het resultaat van het ontwerpproces (een informatiesysteemontwerp) van een hogere kwaliteit zijn.
Het achterhalen en inventariseren van constraints is geen eenvoudige zaak. Door vragen te stellen als:
- hoe wordt een tot een bepaald objecttype behorend object geïdentificeerd?
- welke waarden mag een bepaald attribuut aannemen?
- moet een bepaald attribuut altijd aanwezig zijn?
- welke verzamelingen van objecten behorende tot verschillende objecttypen zijn mogelijk?

kan enig inzicht worden verkregen in de voor een bepaalde situatie geldende beperkingen. Voorts kunnen de volgende twee indelingen hierbij van nut zijn:
1. In het algemeen kan men een onderscheid maken tussen *statische* en *dynamische* constraints ([Remm82], [Griet82]). Statische constraints hebben betrekking op de toestand van een database op een bepaald moment; dynamische constraints daarentegen zijn uitspraken over toegestane overgangen van objecten (transition rules).
2. Verder kan men constraints nog onderverdelen in constraints die betrekking hebben op
   - elke individuele occurrence behorend tot een bepaald objecttype
   - de verzameling van de tot een bepaald objecttype behorende occurrences
   - tot verschillende objecttypen behorende occurrences.

Door deze beide indelingen te combineren kan men een zestal klassen constraints onderscheiden, een raamwerk dat een aardig hulpmiddel kan zijn bij het inventariseren van de constraints die bij de verschillende in het gegevensmodel onderscheiden objecttypen behoren.
We zullen de diverse voor het in beschouwing genomen objectsysteem (de ontwerpmethodiek MOS) geldende constraints hier niet weergeven, doch volstaan met de navolgende constateringen. Een nadere beschouwing van deze regels leert dat er twee *soorten* constraints bestaan. Sommige constraints kunnen en moeten direkt bekrachtigd worden om een tweetal redenen: de File-Handler moet naar behoren kunnen functioneren en uit het oogpunt van gebruiksvriendelijkheid is een direkte respons aanbevelenswaardig [Shnei80]. Deze categorie constraints wordt verder 'categorie 1' genoemd. Een tweede categorie constraints ('categorie 2') wordt gevormd door die constraints waaraan niet noodzakelijkerwijs direkt voldaan hoeft te zijn. In eerste instantie worden deze constraints zelfs altijd overtreden. Hierbij moet men met name denken aan constraints die betrekking hebben op *schema's*. Een diagram moet aansluiten op diagrammen op hetzelfde en op andere abstractieniveaus, en voor de inhoud en opbouw van een diagram gelden vele richtlijnen en regels. Ook al vanuit het oogpunt van gebruiksvriendelijkheid bezien zal het meta-informatiesysteem de gebruiker dus de mogelijkheid moeten bieden om (tijdelijk) meerdere nog inconsistente of incomplete schema's op te slaan. Een dergelijke werkwijze lijkt echter in flagrante tegenspraak met de eerder in deze paragraaf geformuleerde eis dat de integriteit van de database nimmer geweld mag worden aangedaan.
Toch kan aan beide conflicterende eisen voldaan worden, zoals moge blijken uit de volgende constructie: een schema krijgt een attribuut status. Dit attribuut heeft als domein de waarden 'voorgesteld' en 'geaccepteerd'. Met dit attribuut wordt aangegeven dat er twee 'soorten' schema's bestaan. De constraints die tot categorie 1 behoren gelden voor beide soorten schema's. De tot categorie 2 behorende constraints gelden echter alleen voor 'geaccepteerde' schema's. Op deze wijze is de database nooit inconsistent en is het toch mogelijk om nog incomplete/inconsistente schema's op te slaan. Het subsysteem dat bij elke actie op de database nagaat of aan de diverse tot categorie 1 behorende constraints voldaan is wordt *Enforcer* genoemd. Aangezien elke actie op de database de Enforcer moet passeren is dit subsysteem in figuur 1 weergegeven als een 'schil' om de File-Handler. Elk nieuw schema krijgt de status 'voorgesteld'. Wijzigingen in een 'voorgesteld' schema hebben geen gevolgen voor de status van het diagram.
Het subsysteem dat nagaat of de diverse tot categorie 2 behorende regels wel nageleefd worden wordt *Analyser* genoemd. Op het moment dat de gebruiker wil weten of een schema voldoet aan alle regels, en daarbij in overeenstemming is met de reeds door de Analyser gekontroleerde inhoud van de database (alle 'geaccepteerde' schema's) geeft hij de Analyser opdracht het schema te analyseren. Wanneer er geen constraints overtreden blijken te zijn wijzigt dit subsysteem de status van het schema in 'geaccepteerd'. In alle overige gevallen produceert de Analyser een overzicht van alle inconsistenties en onvolledigheden. De informatie-analist kan natuurlijk ook een reeds door de Analyser geaccepteerd schema opnieuw in bewerking nemen. Elke verandering in een dergelijk schema (met uitzondering van een wijziging van een niet-sleutel attribuut) heeft echter tot gevolg dat de integriteit van de database niet langer meer gewaarborgd is. Elke wijziging heeft dan ook tot gevolg dat de Enforcer de status van een schema verandert in 'voorgesteld'. Wanneer het schema 'hoog in de schemaboom' gewijzigd wordt bestaat er natuurlijk een gerede kans dat er in afstammelingen van dat schema inconsistenties optreden. Elke door de wijziging mogelijkerwijs beïnvloed schema krijgt dan ook de status 'voorgesteld'. De informatie-analist zal vervolgens de gehele boom van schema's moeten doorlopen met de Analyser. Dit kan in géén geval automatisch: een *mens* zal moeten bepalen of elk schema ook *inhoudelijk* gezien nog klopt. Schema's die aan alle constraints voldoen kunnen tenslotte inhoudelijk gezien bol staan van de onjuistheden!
Met de in het voorafgaande beschreven werkwijze is een grote mate van flexibiliteit en een juiste balans tussen formaliteit en gebruiksvriendelijkheid geïntroduceerd. Elke vergroting van de mate van vrijheid bij het hanteren van een informatiesysteem vergroot echter tevens de mate waarin – zij het opzettelijk, zij het onopzettelijk – 'misbruik' van dat systeem kan worden gemaakt. In het onderhavige geval zal een werkelijke vergroting van de effectiviteit van het ontwerpproces slechts bewerkstelligd kunnen worden wanneer telkens getracht wordt om zo mogelijk met een volledig consistente en complete (dus door de Analyser geanalyseerde en akkoord bevonden) database te werken. De informatie-analist zal zich moeten realiseren dat een grote hoeveelheid 'voorlopige' schema's bepaald niet aanbevelenswaardig is; een voorlopig schema dient van tijdelijke aard te zijn. Het systeem kan een dergelijke zienswijze bevorderen door bijvoorbeeld aan de start van elke sessie een lijst met voorlopige en nog te analyseren schema's te produceren, of door bij overschrijding van een bepaald aantal van dergelijke schema's alleen nog bewerkingen op die schema's te accepteren.

Figuur 3: Scherm-layout bij gebruik van de Grafische Interface



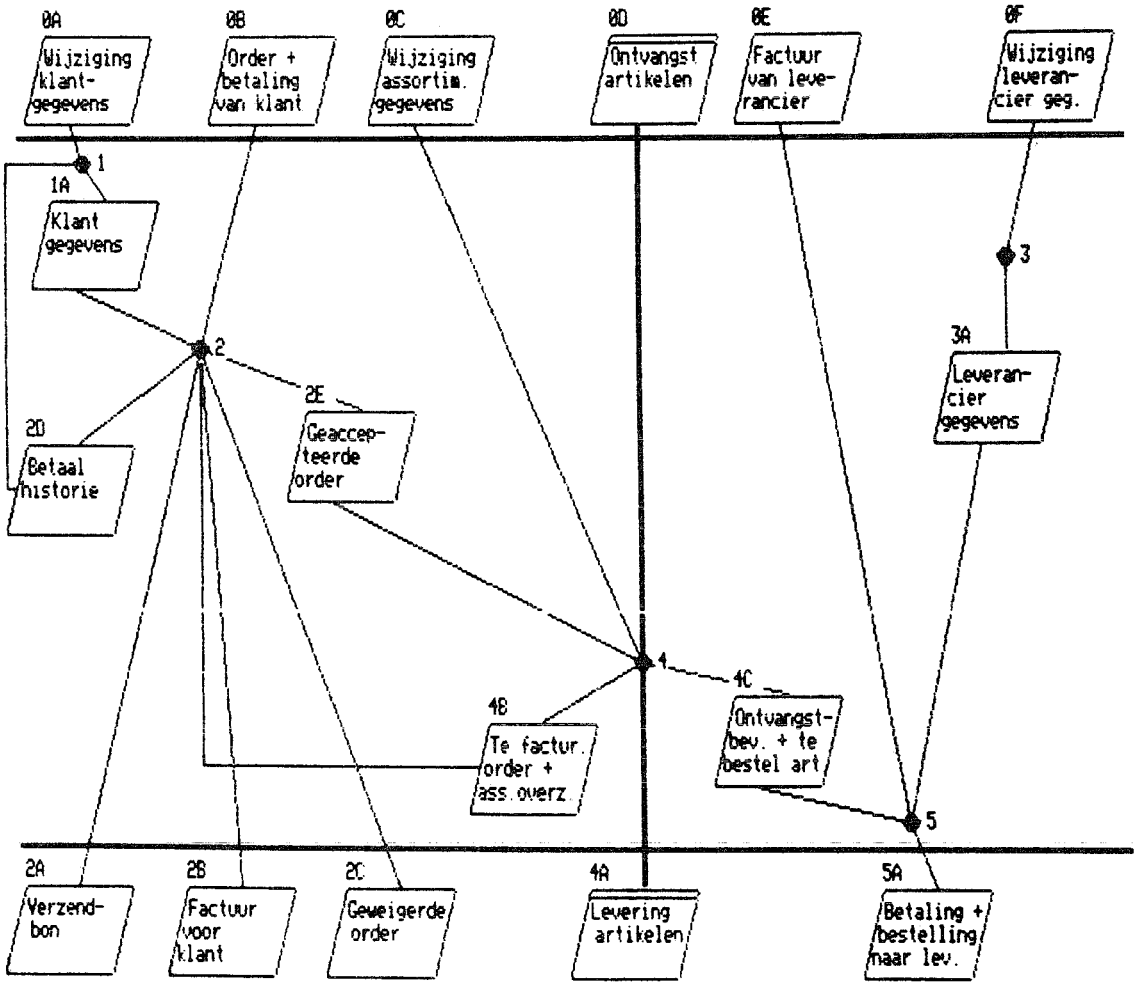
5.2.3 *De Grafische Interface*

De *Grafische Interface* is één van de twee interfaces via welke de informatie-analist of systeemontwerper met het systeem communiceert. Via deze Interface kunnen schema's op het scherm worden opgebouwd en gewijzigd, waarna eventueel opslag in de database kan volgen. De gebruiker communiceert met het systeem door met een 'touchpen' een gewenste actie te selecteren uit de rechts op het scherm verschijnende menu's. Figuur 3 bevat een gestileerde weergave van de beeldscherm-layout.

De menu's worden geplaatst in kolom 70 t/m 80 van het beeldscherm. De onderste regel van het scherm wordt gebruikt voor aanwijzingen van het systeem aan de gebruiker, vragen om additionele invoer, en foutmeldingen (die doorgaans op het tweede gedeelte van die regel verschijnen, zodat er ruimte blijft bestaan voor verdere aanwijzingen van het systeem aan de gebruiker ervan). Het resterende gedeelte van het scherm is bestemd voor het weergeven van schema's. Deze ruimte wordt tevens benut voor HELP-informatie: de schema's worden dan (tijdelijk) gewist. De in de menu's beschikbare commando's hebben voornamelijk betrekking op het ophalen, opslaan en afdrukken (zie voor een dergelijke afdruk figuur 4) van schema's, en op het toevoegen, verwijderen en verplaatsen van elementen in die schema's. De touchpen wordt verder gehanteerd voor het aanwijzen of

Figuur 4: Grafische output via de printer

SCHEMACODE: A0	VERSIE: 1	DATUM: 28/2/84	AUTEUR: RVo
SCHEMANAAM: A0 - Huidige Situatie		PROJEKT:	



plaatsen van een element. Wanneer de informatie-analist een nieuw of bewerkt schema wil opslaan wordt de gedurende de sessie in het interne geheugen opgebouwde representatie van het schema vergeleken met de inhoud van de database, waarna alle veranderingen doorgevoerd worden. Het blijft dus mogelijk om de database ongewijzigd te laten wanneer het resultaat van een sessie bij nader inzien onbevredigend is. Bij het opbouwen van schema's blijft het de taak van de informatie-analist om te zorgen voor een nette schema-layout; het systeem neemt, voornamelijk om de responsietijden laag te houden, deze taak niet over. Tekst-invoer tenslotte geschiedt, wanneer nodig, via het toetsenbord.
De via de Grafische Interface gegeven commando's hebben altijd betrekking op één schema tegelijkertijd. Ingevoerde informatie is altijd direkt nodig voor het opbouwen c.q. wijzigen van dat schema, en heeft altijd direkt zichtbare gevolgen voor de opbouw van het onderhavige schema. Natuurlijk kunnen via deze Interface in een schema aangebrachte wijzigingen wel degelijk gevolgen hebben voor (zeer grote aantallen) andere schema's. Dit verschijnsel hangt samen met de bewaking van de integriteit van de database, waarover uitgebreid gesproken is in de vorige paragraaf.
Bij de ontwikkeling van het prototype van dit subsysteem is veel aandacht aan de gebruiksvriendelijkheid geschonken. In het navolgende zal dit met behulp van enkele voorbeelden worden geïllustreerd.

- *Interactiemechanisme*
  De gebruiker maakt gewenste acties kenbaar aan het systeem door met een touchpen te kiezen uit een menu. Deze wijze van interactie is om een aantal redenen verkozen boven andere mogelijke interactiemechanismen als functietoetsen of commandotalen. Een eerste belangrijke overweging is van ergonomische aard. Door gebruik te maken van aanraakmenu's wordt het aantal malen dat de informatie-analist van invoermedium moet wisselen geminimaliseerd. De gebruiker kan zijn aandacht op het beeldscherm gevestigd houden; het merendeel der acties vereist in het geheel geen invoer via het toetsenbord meer. Wanneer gebruik gemaakt zou zijn van een commandotaal zou de gewenste operatie telkens via het toetsenbord dienen te worden gespecificeerd, waarna doorgaans de touchpen ter hand zou moeten worden genomen om coördinatenparen in te voeren. Ten tweede hebben menu's in het algemeen het voordeel dat het geheugen van de gebruiker wordt ondersteund ([Shnei80], [Oost83]). Deze wordt op elk moment geconfronteerd met een aantal mogelijke acties, en hoeft tevens niet na te denken over het commando dat – of de functietoets die – bij een gewenste actie hoort. Een algemeen nadeel van menu's – het feit dat ze naarmate de ervaring met een systeem groter wordt vervelend kunnen worden – geldt niet voor de menu's in de Grafische Interface. Deze irritatie wordt namelijk o.a. veroorzaakt doordat menu's vaak op een hiërarchische wijze worden opgebouwd. Om één actie te selecteren dient de gebruiker uit meerdere menu's zijn keuze te maken. Dit geldt echter niet voor de Grafische Interface: de menu's weerspiegelen hier de werkwijze van de informatie-analist bij het opbouwen en wijzigen van schema's.
- *Foutafvanging en herstelmogelijkheden*
  Een informatie-analist kan bij het opbouwen van schema's verschillende soorten fouten maken. Fouten dienen om evidente redenen zo mogelijk voorkomen te worden; in geval de informatie-analist een vergissing maakt die door het systeem niet als zodanig herkend kan worden moet de mogelijkheid bestaan deze vergissing op een eenvoudige wijze te herstellen. Verder moeten gemaakte en voor het systeem herkenbare fouten resulteren in een duidelijke foutboodschap.
- *Help-functie*
  Met name de beginnende gebruiker zal uitgebreide informatie wensen over de hem ter beschikking staande functies, en de te ondernemen actie wanneer er problemen optreden. Het systeem zal daarom over een uitgebreide HELP-functie dienen te beschikken. Via de HELP-toets in de diverse menu's zal ten eerste algemene informatie over het systeem beschikbaar moeten komen. Ten tweede zal er gedetailleerde informatie m.b.t. een bepaalde functie verkregen moeten kunnen worden. Dergelijke informatie kan bijvoorbeeld op het scherm geplaatst worden na het achtereenvolgens aanraken van de HELP-toets en de toets (functie) waarover men HELP-informatie wenst. Ten derde zal er, wanneer HELP aangeraakt wordt na een foutboodschap, uitgebreide informatie over de gemaakte fout (en de herstelmogelijkheden) op het scherm moeten verschijnen. Aangezien de ruimte voor foutboodschappen in het onderhavige geval beperkt is, en deze daarom kort en soms onduidelijk zullen zijn, is een dergelijke faciliteit noodzaak.

De gebruiksvriendelijkheid van een systeem wordt natuurlijk door meer aspecten dan de diverse in het voorafgaande besproken factoren bepaald. Zo is het bijvoorbeeld bijzonder belangrijk dat aan de zgn. prestatie-eisen voldaan wordt. We zullen daar echter in dit artikel niet nader op ingaan.

5.2.4 *Overige subsystemen*
In deze paragraaf zullen de subsystemen 'Tekstuele Interface' en 'Productie Afleidbare Overzichten' kort behandeld worden.
De *Tekstuele Interface* verzorgt het merendeel van de één-dimensionale (tekstuele) communicatie met de gebruiker. Door niet elk subsysteem zelf de vereiste communicatie tussen gebruiker en systeem te laten verzorgen wordt het creëren van een uniform uiterlijk sterk vereenvoudigd. Tevens wordt het mogelijk om bij het ontwerp van de overige subsystemen (natuurlijk met uitzondering van de Grafische Interface) grotendeels te abstraheren van aspecten met betrekking tot de gebruiksvriendelijkheid en zich te concentreren op de eigenlijke functie van het subsysteem.
Via de Tekstuele Interface kunnen beschrijvingen van schema's worden ingevoerd, gewijzigd en opgeslagen. Op deze wijze ingevoerde beschrijvingen kunnen met de Grafische Interface – op interactieve wijze – omgezet worden in schema's; omgekeerd is het ook mogelijk om beschrijvingen van de via de Grafische Interface opgebouwde schema's met de Tekstuele Interface te wijzigen. Via dit subsysteem kan het in ontwikkeling zijnde informatiesysteem gedetailleerder beschreven worden dan via de Grafische Interface: ook synoniemen, beschrijvin-

gen, opmerkingen, etc. kunnen via deze Interface worden opgeslagen in de database. De Tekstuele Interface zal een groot aantal vrij krachtige commando's moeten kunnen herkennen en uitvoeren. Ten eerste zal de gebruiker willen beschikken over een krachtige query-faciliteit. Ook op ad-hoc vragen zal een antwoord moeten kunnen worden verkregen. Dit impliceert dat de gebruiker zal moeten beschikken over een query-taal waarmee zowel vooraf gedefinieerde en gecompileerde als on-line queries kunnen worden uitgevoerd. Ten tweede zal de gebruiker de beschikking willen hebben over krachtiger operaties dan operaties die slechts betrekking hebben op één element in de database. De Tekstuele Interface zal verschillende soorten commandotalen moeten ondersteunen, zodat gebruikers ongeacht hun ervaring met het systeem, moeiteloos ermee kunnen communiceren. Daar beginnende gebruikers behoefte zullen hebben aan een strakkere, meer gedetailleerde leidraad dan de meer ervaren gebruikers, zullen deze commandotalen verschillen in complexiteit. Voor de onervaren gebruiker zal er een mogelijkheid moeten bestaan om via menu's wensen aan het systeem kenbaar te maken. Omdat via de Tekstuele Interface een groot aantal commando's zullen moeten kunnen worden gegeven zullen, teneinde het aantal keuzen per menu niet al te groot te maken, menu's veelal moeten worden opgesplitst in een hiërarchische verzameling submenu's. Het telkens opnieuw doorlopen van de hele 'menu-boom' zal vervelender worden naarmate de ervaring met het systeem toeneemt. Dit geldt met name wanneer deze menu's slechts langzaam worden opgebouwd. De geoefende gebruiker zal dan ook over de mogelijkheid moeten kunnen beschikken om de menufaciliteit naar behoeven aan/uit te schakelen. Voor deze categorie gebruikers zal er een commandotaal en een beschrijvingstaal gecreëerd moeten worden. Hierbij moet dan weer de mogelijkheid bestaan om met afkortingen te werken.

Het subsysteem *Produktie Afleidbare Overzichten* tenslotte heeft tot taak allerhande afleidbare specificaties te genereren. Hierbij kan men bijvoorbeeld denken aan een gevisualiseerd overzicht van de hiërarchische verbanden tussen de in de database aanwezige schema's, aan schema's die een rol spelen in de technisch ontwerpfase, of aan verschillende cross-references. Dit subsysteem neemt aldus diverse routinematige taken over van de informatie-analist. In tegenstelling tot de Tekstuele Interface, via welke voornamelijk selecties uit de in de database aanwezige ontwerpgegevens kunnen worden verkregen, voegt 'Productie Afleidbare Overzichten' iets toe aan deze gegevens.

5.3 Status en Ontwikkelingen

De huidige status van het project is als volgt. Er bestaat momenteel een operationeel prototype van een tweetal modules: de File-Handler en de Grafische Interface. Hiermee kan een grafische representatie van een te ontwikkelen informatiesysteem worden opgebouwd, gewijzigd, opgeslagen in en verwijderd uit de 'design-database'. Verder kan men reeds beschikken over een bescheiden voorloper van de Tekstuele Interface, waarmee – via een beperkt aantal, eenvoudige, commando's – de inhoud van de database gewijzigd kan worden. De in paragraaf 5.2.2 beschreven Enforcer is gedeeltelijk gerealiseerd; de eerste versie van de Analyser moet nog gebouwd worden. Aan het subsysteem Productie Afleidbare Overzichten is nog geen aandacht besteed.

Het tot dusverre ontwikkelde prototype kan momenteel reeds van praktisch nut zijn voor de met de ondersteunde methode werkende informatie-analist. Met dit hulpmiddel behoort het tijdrovende tekenwerk reeds tot het verleden. Omdat schema's bijzonder snel kunnen worden gecorrigeerd zal de informatie-analist minder de neiging vertonen om het documenteren zo lang mogelijk uit te stellen (teneinde de inspanning welke gepaard gaat met het onderhoud van de ontwikkelingdocumentatie te verminderen): de documentatie zal meer up-to-date zijn. Het huidige prototype kan nu al een belangrijke stimulans voor een verbetering van de communicatie tussen informatie-analist en eindgebruikers betekenen. Deze kunnen gezamenlijk achter het scherm gezeten snel alternatieve constructies doornemen, totdat een voor beiden bevredigend resultaat is ontstaan. Vanwege het feit dat interactieve grafische systemen voor velen voorlopig nog een vrij nieuw en boeiend verschijnsel vormen bestaat er een gerede kans dat eindgebruikers uiteindelijk gemotiveerd raken om dit gereedschap zelf ter hand te nemen.

In de nabije toekomst zal er een pilot-project moeten worden gestart waarin de ontwikkeldè programmatuur op een aantal punten zal moeten worden geëvalueerd. In een dergelijke beschermde omgeving zal een antwoord moeten worden gevonden op vragen als:

- zijn alle gewenste functies aanwezig;
- blijven responsietijden, ook bij een reële omvang van de database, aanvaardbaar;
- voldoet de huidige apparatuur (is de beperkte schermgrootte aanvaardbaar, is de touchpen een geschikt invoermedium, etc.).

Gedurende het pilot-project kan verder waardevolle informatie betreffende het gebruik van het gereedschap verkregen worden. Men kan bijvoorbeeld de frequentie waarmee de verschillende functies gebruikt worden laten bijhouden.

Parallel aan deze pilot zal het prototype verder uitgebreid moeten worden. Nieuwe subsystemen zullen in het pilot-project geëvalueerd moeten worden. Na een dergelijke grondige evaluatie zal overgegaan moeten worden tot ontwikkeling van het definitieve systeem. Op dat moment is er sprake van een volwaardig product, dat echter nog steeds voortdurend uitgebreid zal worden. Hierbij wordt bijvoorbeeld gedacht aan het incorporeren van faciliteiten ten behoeve van het projectmanagement. Verder zal het systeem geschikt gemaakt moeten worden voor een multi-user omgeving. Tenslotte zullen methodieken als MOS met name gehanteerd moeten worden in grote complexe projecten waarin men zonder een methodische aanpak al snel het overzicht zou verliezen. Dergelijke projecten worden typisch gerealiseerd door een team, bestaande uit meerdere informatie-analisten, systeemontwerpers en programmeurs. In een dergelijke omgeving zal men er toe moeten overgaan om een centrale data-dictionary en/of database te hanteren. Alle leden van het ontwerpteam zouden dan toegang moeten hebben tot deze data-dictionary. Wanneer subsystemen door verschillende informatie-analisten worden ontwikkeld zullen er faciliteiten moeten bestaan om de deeloplossingen te integreren. In principe kan men hiervoor een subsysteem gelijk de Analyser ontwikkelen: men heeft echter wel te maken met meerdere gebruikers. Er

zal dan dus aandacht besteed moeten worden aan het vastleggen van verantwoordelijkheden en bevoegdheden. Teneinde niet te zeer afhankelijk te worden van de in een specifieke bedrijfssituatie beschikbare programmatuur (data-dictionary manager/database management system) zullen er interfaces ontwikkeld moeten worden via welke de in de lokale 'design-database' aanwezige ontwerpinformatie overgeheveld zou kunnen worden naar de centrale database. Aanpassing van het systeem aan een nieuwe omgeving zal dan voornamelijk beperkt kunnen blijven tot aanpassing van de interface.
Het ligt in de bedoeling het in de voorgaande paragrafen beschreven gereedschap eind 1985 gerealiseerd te hebben. Het verwezenlijken van dit gereedschap moet gezien worden als een eerste stap in de richting van een geïntegreerd, het gehele informatiesysteemontwikkelingstraject ondersteunend meta-informatiesysteem.

6 CONCLUSIES

De informatie-analist/systeemontwerper wordt bij het hanteren van een als procesbenadering te karakteriseren methode geconfronteerd met het probleem de ontstane documentatie te onderhouden. Dit vergt zoveel tijd dat het gevaar dreigt dat niet het analyseren van probleemsituaties maar het juist hanteren van de methode primair komt te staan. Projecten (en informatie-analisten) dreigen ten onder te gaan in een zee van papier. Een geautomatiseerd hulpmiddel zal de productiviteit van de informatie-analist/systeemontwerper kunnen verhogen. Bovendien zal het functionele informatiesysteemontwerp, doordat de informatie-analist of systeemontwerper wordt ondersteund bij het handhaven van de consistentie en volledigheid van de grote hoeveelheden gedurende het analyse- en ontwerpproces geproduceerde informatie, van een significant hogere kwaliteit zijn. De systeemdocumentatie zal up-to-date zijn, doordat de informatie-analist/systeemontwerper welhaast gedwongen wordt direkt te documenteren. Een groot deel van dit artikel is gewijd geweest aan de beschrijving van een dergelijk hulpmiddel, een gereedschap dat kan worden gekarakteriseerd als een meta-informatiesysteem, ontwikkeld als ondersteuning voor de ontwerpmethodiek MOS.
Het concept laat zich als volgt omschrijven. Centraal staat de verzameling opgeslagen analyse- en ontwerpinformatie ('design-database'). Er wordt een sterke nadruk gelegd op het handhaven van de integriteit van deze database. Wanneer een gebruiker wijzigingen aanbrengt worden onduidelijkheden, inconsistenties en onvolledigheden op interactieve wijze opgelost. De gebruiker wordt zo snel mogelijk geattendeerd op een fout of een overtreding van de binnen de ontwerpmethode geldende regels. Bepaalde constraints kunnen – alleen al uit het oogpunt van gebruiksvriendelijkheid – niet direkt bekrachtigd worden. Of de tot deze categorie behorende constraints overtreden zijn wordt nagegaan op verzoek van de gebruiker. De gekozen constructie biedt de gebruiker de mogelijkheid om nog niet volledige of nog inconsistente schema's – tijdelijk – op te slaan, zonder dat de integriteit van de database geweld wordt aangedaan.
Met het tot dusverre gerealiseerde systeem kan het in ontwikkeling zijnde informatiesysteem op een tweetal wijzen beschreven worden: er kan zowel een grafische als een tekstuele representatie worden opgebouwd. Schema's kunnen op een interactieve wijze worden geconstrueerd; de informatie-analist hanteert hierbij een touchpen om commando's te selecteren uit menu's, en om elementen in schema's te plaatsen of aan te wijzen. De informatie-analist kan echter ook een tekstuele beschrijving van het informatiesysteem in wording opbouwen, en dus geen gebruik maken van de grafische faciliteiten. Uit de op deze wijze ingevoerde ontwerpinformatie kunnen indien gewenst later op interactieve wijze schema's worden gegenereerd. Met een aantal vooraf gedefinieerde queries kunnen diverse overzichten van de in de database aanwezige ontwerp-informatie worden verkregen. Voorts kunnen allerhande afleidbare (reeds impliciet aanwezige) specificaties gegenereerd worden. Het systeem wordt gerealiseerd op een micro-computer, en moet eind 1985 geheel operationeel zijn. Een uitgebreide beschrijving van het gereedschap kunt u vinden in [Vonk84a].
Een systeem als in dit artikel beschreven kan men bouwen voor elke top-down georiënteerde procesbenadering die een sterke nadruk op de grafische beschrijvingswijze legt (denk bijv. aan SADT, SASO, TIA, ISAC, MOS, SA, PRISMA). Na een gegevens-analyse van de desbetreffende methode kan een 'design-database' worden opgebouwd. Via een subsysteem ongeveer gelijk aan de Grafische Interface zouden diagrammen kunnen worden opgebouwd. Natuurlijk zou het systeem de gebruiker ook de mogelijkheid moeten bieden om tekstuele beschrijvingen in te voeren en allerhande overzichten van de inhoud van de database te produceren (Tekstuele Interface). De regels die voor de in beschouwing genomen methodiek gelden zouden bekrachtigd kunnen worden door een component als de Analyser. Merk overigens op dat de voor de diverse genoemde methodieken geldende constraints niet fundamenteel en soms geheel niet van elkaar verschillen. Al tijdens de realisatie van een dergelijke tool zal men aandacht moeten schenken aan het creëren van mogelijkheden om te komen tot een integratie met gereedschappen voor latere fasen van het informatiesysteemontwikkelingsproject.
Ter relativering van de rol van methoden en technieken is al diverse malen geconstateerd dat het de ontwerper is die ontwerpt, en niet de methode [Bemel81]. Hetzelfde geldt uiteraard voor een, de methode ondersteunend, geautomatiseerd hulpmiddel: een onervaren ontwerper zal zelfs met uitstekend gereedschap geen goed informatiesysteemontwerp kunnen produceren. Daartegenover staat dat een geautomatiseerde ondersteuning als in dit artikel beschreven het juist hanteren van de methode niet alleen mogelijk maakt, maar dit zelfs zal kunnen afdwingen. Ofschoon semantische aspecten vooralsnog minder aan bod komen, zal gebruik van een dergelijk hulpmiddel resulteren in een syntactisch juist en volledig functioneel ontwerp: een resultaat dat aandacht voor deze materie alleszins rechtvaardigt.

7 LITERATUUR

[Baber82] Baber, *R. L., Software Reflected: The Socially Responsible Programming of Our Computers*, North-Holland Publishing Company, Amsterdam New York/Oxford, 1982.

[Beek82] Beek, J van, 'Een systeemontwikkelingsmethode gebaseerd op prototyping'. In: *Informatie*, jrg. 24, nr. 12, dec. 1982, pag. 702-710.

[Bell77] Bell, T E., Bixler, D. C., Dyer, M. E., 'An extendable approach to computer-aided software re-

quirements engineering'. In: *IEEE Transactions on Software Engineering*, Vol. SE-3, nr 1, jan. 1977, pag. 49-59

[Bemel81] Bemelmans, T. M. A., Boer, J. G. de, 'Ontwikkelingsmethoden 1: Het ontwikkelen van informatiesystemen'. In: *Informatie*, jrg. 23, nr. 2, feb. 1981, pag. 67-75.

[Bemel83a] Bemelmans, T. M. A., 'Ontwikkelingsmethoden voor informatiesystemen: onopgeloste problemen'. In: *Informatie*, jrg. 25, nr. 2, feb. 1983, pag. 6-11

[Bemel83b] Bemelmans, T. M. A., 'Empirisch onderzoek over informatiesystemen: Verslag van een conferentie'. In: *Informatie*, jrg. 25, nr. 2, feb. 1983, pag. 39-50.

[Boer84] Boer, J. G. de, 'Het ontwikkelen van geautomatiseerde informatiesystemen in organisaties en de rol daarin van methodieken'. In: *Bedrijfskunde*, jrg. 56, nr. 1, 1984, pag. 9-18.

[CDC82] Control Data Corporation, *Information Analysis Support Tools Reference Manual*, CDC, jan. 1982.

[Davis77] Davis, C. G., Vick, C. R., 'The Software Development System'. In: *IEEE Transactions on Software Engineering*, vol. SE-3, nr. 1, jan. 1977

[Davis82] Davis, G. B., 'Strategies for information requirements determination'. In: *IBM Systems Journal*, Vol. 21, nr. 1, 1982, pag. 4-30

[Delis82] Delisle, N. M., Menicosy, D. E., Kerth, N. C., 'Tools for Supporting Structured Analysis'. In [Schnei82].

[Dieb82] The Diebold Group, Inc., '*The Diebold Information Technology Scan*, 1982', The Diebold Group, Inc., Document Number 209T, Technology Series, 1982.

[Gerst81] Gersteling, H., Schotgerrits, A. H. J. B., 'Ontwikkelingsmethoden 13: Het ISDOS-project en PSL/PSA'. In: *Informatie*, jrg. 23, nr. 9, sept. 1981, pag. 515-531.

[Gerst83] Gersteling, H., '*Evaluatierapport PSL/PSA*', Intern rapport CVI, Utrecht, sept. 1983

[Gremi83] Gremillion, L. L., Pyburn, P., 'Breaking the systems development bottleneck'. In: *Harvard Business Review*, March-April 1983, pag. 130-137.

[Griet82] Griethuysen, J. J. van, (editor), '*Concepts and Terminology for the Conceptual Schema and the Information Base*', ISO TC97/SC5/WG3, maart 1982.

[Griet84] Griethuysen, J. J. van, Jardine, D. A., 'De INFOMOD-benadering van informatiemodellering'. In: *Informatie*, jrg. 26, nr. 6, juni 1984, pag. 423-443.

[Haus82] Hausen, H. L., Mullerburg, M., '*Software Engineering Environments: State of the art, problems and perspectives*', Proc. IEEE Compsac 82, IEEE Computer Society Press, Silver Spring, 1982.

[Hecht82] Hecht, H., Houghton, R. C., '*The current status of software tool usage*', Proc. IEEE Compsac 82, IEEE Computer Society Press, Silver Spring, 1982.

[Hend82] Henderson, P., *System Design*, Infotech State of the Art Report, series 9, nr. 6, Pergamon Infotech 1982.

[Hrus83] Hruschka, P., '*The software engineering environment PROMOD*', in Proc. ESA/ESTEC Software Engineering Seminar, Noordwijk, okt. 1983, pag. 59-63.

[Koba83] Kobayashi, M., Nogi, K., Kataoka, M., Hayashi, T., Aoyama, Y., Mitsumaki, T., '*ICAS: An Integrated Computer Aided Software Engineering System*', Proc. IEEE Compcon 83, IEEE Computer Society Press, Silver Spring, 1983.

[Lissa84] Lissandre, M., Lagier, P., Skalli, A., Massie, M., '*Specif: a Specification Assistance System*', Manual SPECIF, IGL, 1984.

[Nassi80] Nassi, I., 'A Critical Look at the Process of Tool Development: An Industrial Perspective'. In [Ridd80].

[NGI83] Methodieken voor Informatiesysteemontwikkeling (uitgebreide uitgave), serie ontwikkelingsmethoden uit het maandblad *Informatie*, NGI Rapport nr. 3a, Amsterdam, Nederlands Genootschap voor Informatica, maart 1983.

[Oost83] Oostrum, P. van, 'Interactie met persoonlijke informatiesystemen'. In: *Informatie*, jrg. 25, nr. 11, nov. 1983, pag. 50-62.

[Remm82] Remmen, F., '*Databases: grondslagen voor de logische structuur*', Academic Service, Den Haag, 1982.

[Ridd80] Riddle, W. E., Fairly, R. E. (editors), '*Software Development Tools*', Springer-Verlag, Berlijn, 1980.

[Schel81] Schell, H. J., 'Ontwikkelingsmethoden 10: Systematrix (SMX)'. In: *Informatie*, jrg. 23, nr. 6, juni 1981, pag. 361-369.

[Schnei82] Schneider, H. J., Wasserman, A. I., (editors), '*Automated Tools for Information Systems Design*', North Holland Publishing Company, Amsterdam, 1982.

[Shnei80] Shneiderman, B., '*Software Psychology: Human Factors in Computer and Information Systems*', Winthrop Publishers, inc., Cambridge, Massachusetts, 1980.

[Smith80] Smith, G. L., Stephens, S. A., Tripp, L. L., Warren, W. L., '*Incorporating Usability into Requirement Engineering Tools*', Proc. ACM'80, Nashville, 1980.

[Strau83] Strausz, I., 'De kwaliteit van software'. In: *Informatie*, jrg. 25, nr. 12, dec. 1983, pag. 30-33.

[Teich77] Teichroew, D., Hershey, E. A., 'PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems'. In: *IEEE Transactions on Software Engineering*, Vol. SE-3, nr. 1, jan. 1977.

[Vonk84a] Vonk, R., '*COMOS: concept en prototype van een geautomatiseerde ondersteuning voor MOS*'. Afstudeerverslag Technische Hogeschool Eindhoven, feb. 1984.

[Vonk84b] Vonk, R., '*Prototyping: Concepts and Guidelines*', Interne publicatie Philips International, Corporate ISA, TMS/SM, cat. nr. 4322 270 20451, ref. UDR-TMS/SM-84/115, 1984.

[Wagen81] Wagenaar, W., 'Ontwikkelingsmethoden 18: MOS: Methodisch Ontwikkelen van Systemen'. In: *Informatie*, jrg. 23, nr. 12, dec. 1981, pag. 778-783.

[Wass82a] Wasserman, A. I., '*Software Tools and the User Software Engineering Projekt*'. In [Ridd80].

[Wass82b] Wasserman, A. I., '*Towards integrated software development environments*'. In [Hend82].