Rijksuniversiteit Groningen **Faculteit der Wiskunde en Natuurwetenschappen**

**Vakgroep Informatica**





# ITS MAGIC
# Design and Implementation of an Intelligent Interactive Tele–Shopping Application for the Electronic Highway

R. Buist
J. de Graaf
W. Wichers

begeleider: Prof.dr.ir. L. Spaanenburg

augustus 1996

RuG

# Contents

# Abstract

A framework has been created to prototype interactive, personalized commercial processes in a distributed multi–media environment such as the Electronic Highway. It exploits a number of new technologies to obtain a highly generic software package. The report documents all the design phases from system requirement capture to intelligent motion planning. The result is a highly structured, adaptive, object–oriented composition of re–usable, trainable and overlayable segments, emphasizing portability and maintainability. The software package comes with an Implementators Guide and a Function Reference Manual.



# Samenvatting

Er is een raamwerk ontwikkeld waarmee interactieve, persoonlijke commerciële processen gedemonstreerd kunnen worden in een gedistribueerde multi–media omgeving zoals de Elektronische Snelweg. Het raamwerk maakt gebruik van een aantal nieuwe technieken waardoor een uiterst generiek software pakket verkregen is. In het verslag worden alle ontwikkelingsfasen beschreven, van het vastleggen van de system requirements tot de intelligent motion planning. Het resultaat is uiterst gestruktureerd, adaptief, object georiënteerde samenstelling van herbruikbare, trainbare en overlayable segmenten, waarbij de nadruk gelegd wordt op portabiliteit en onderhoudbaarheid. Bij het software pakket wordt een Implementators Guide en een Function Reference Manual bijgeleverd.

# Preface

The Electronic Highway and the World Wide Web with its HTML pages are often seen as synonyms. However, there are many other options for the real Electronic Highway. While politicians are thinking of commericial uses like pay–per–view and other passive amusement options, we feel that more emphasis is needed on interactive commercial applications.

This report, which is a master's thesis, addresses the design phases and the development of a prototype of the ITS MAGIC system, which is a playfully entertaining shopping application for the Electronic Highway. The prototype shows the possibilities of using multi–media and soft computing.

The first chapter is an introduction to the key concepts of ITS MAGIC. The second chapter deals with the economic issues of using the system. These two chapters were used by our professor, Ben Spaanenburg, to convince Dutch companies of the advantages of the ITS MAGIC system, and were therefore written in Dutch. Due to the limited amount of time available, these chapters were not completely translated into English for this thesis.

In chapter 3, the formal specification for the ITS MAGIC system will be described. At the end of that chapter, we will motivate our decision to implement the user part of the system, which is called the K3 User Program. The implementation aspects and the basic system architecture are addressed in chapter 4. The following five chapters contain detailed descriptions of the components the K3 User Program is composed of. Finally, in chapter 9 we will present the conclusions and describe the research and development still needed.

Groningen, August 1996

Ron Buist
Jan de Graaf
Wim Wichers

# Chapter 1 The concept of MAGIC

The project proposal assumes the continued integration of multimedia facilities within hardware platforms. After the increase in pure processing power in the first generations of processors, in the course of which functionality was added per application by means of add on cards, a special component (i.e. the Digital Signal Processor) was developed for this extra skill. With the growing popularity of multimedia applications based on digital signal processors in mind, attempts are being made to implement this functionality on conventional processors. This will lead to the situation in which the unpacking, coloring and playing of audio and video can take place on the user's personal computer (according to his personal characteristics). This increases the possibilities for local individualization, and at the same time it reduces network requirements. The kind of network traffic such a system produces directly matches contemporary considerations regarding network safety.

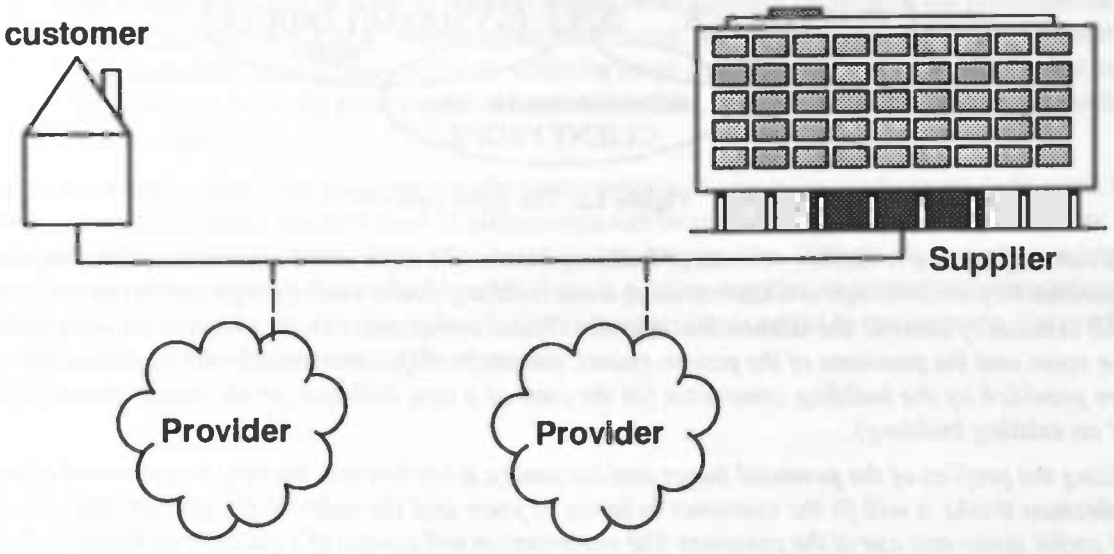



**Figure 1.1:** *The electronic highway*

## 1.1 Starting–points

In this report, a possible framework for an interactive multimedia information system is sketched. The system has to provide information aimed at a specific customer at the time that this is relevant. What are

our goals? The system has to be as user–friendly as possible. This emphasizes the need for non–verbal (and therefore multimedia) communication. Furthermore, a profile should be kept for every user. By means of this profile, a supplier can decide which information and which products are best suited for this particular customer. In this way, he can provide his customers with tailor–made advices and offers.

Changes on the supplier side (like a new product or a new logo) should be perceptible to the customers as soon as possible. The supplier should have the possibility to directly use the reactions given by the customers in his own system. Orders have to be processed, offers sent and packages delivered. Furthermore, other market–parties should have the possibility to use supplier databases to be able to develop new services. An interior decorator, for example, needs to keep in touch with several furniture manufacturers as well as customers in order to be able to offer his services. In general we can assume that the presence of a large amount of varied data can form a stimulus for new products as well as new and changing forms of collaboration.

### 1.1.1 A small illustration

*Let's illustrate this with an example. There are many kinds of furniture elements available for someone wishing to refurnish his kitchen. In order to facilitate the judging, it is useful to select these elements based on the desired function. Possible kitchen furnishing can be visualized by using a floor–plan of the house. In this process, the supplier wants to express the qualities of his products as much as possible. This can be done by further coloring the scene, default selection of specific furniture elements, etc. Finally, there are several individual characteristics which have to be used to produce a suitable scenario. The view at the kitchen, for example, is strongly determined by the length of the potential customer.*


**Figure 1.2**: *The sales process*

*To a manufacturer, a kitchen consists of building blocks like cupboards, sink units and rinsing tubs. The potential buyer can compose a kitchen using these building blocks, taking the properties of the room which will eventually become the kitchen into account. These properties, like for example the measurements of the room and the positions of the power–points, are preferably not entered by the customer. Rather, they are provided by the building contractor (in the case of a new building) or the house–agent (in the case of an existing building).*

*Using the profiles of the potential buyer and his family, a kitchen will initially be presented of which the salesman thinks it will fit the customer in terms of price and the color of the kitchen, but also in terms of social status and age of the customer. The presentation will consist of a guided tour through the kitchen. During this experience of the virtual kitchen the customer can exchange the building blocks he doesn't like with the ones he likes better. There's also a possibility for the customer to walk through the kitchen, during which for example the cupboards are opened to see if they're at the right height.*

*One can come up with several possibilities for the placing of the system. The building contractor and the house–agent were already mentioned, but also a bank or a PTT location should not be excluded. In an all–embracing information–carrying network, as proposed in MAGIC, these will turn out to be early actors on the market. On the long term, the use in a domestic environment is to be expected. State of the art is Albert Heijn's experiment with CD–ROM based shopping and the walk through a house, demonstrated by former Berkeley professor and flower–power guru Timothy Leary.*

### 1.1.2 Basic architecture

The above–mentioned story can be framed in a communication model: an alternative approach to the ongoing discussion on the interaction between Marketing and Sales. In the information–technological concept used here, we see marketing as part of a customer–oriented enterprise, which could, from a diversity in building blocks, eventually lead to sales. Locally, the functionality which has to be developed consists of the following four elements:

- In general, the **product space** consists of the products of one supplier. In the case of a kitchen manufacturer, these will for example be cupboards and sink units; for an insurance company these will be insurance–policies. Elements from different product spaces are presented together (e.g. Bruijnzeel cupboards with a Bosch refrigerator and a Philips built–in radio alarm–clock) in an environment known to the customer.
- The **client space** indicates the environment in which the products are to be used by the customer. This can consist of a floor–plan of the customer's house or situations in the customer's neighborhood. A motor–accident on the highway Assen–Groningen, for example, shown when trying to sell a motor insurance, will be very recognizable for an inhabitant of Vries (Dr.). The elements of the client profile are stored at different organizations, and they have to be presented together and/or after selection.
- The **client profile** contains all the personal information of a customer which can be relevant to a supplier. In this context we don't only think of names and addresses, but also of civil state, favorite color, hobbies, etc. It's important that products are presented in the right setting. For this reason, as a finishing touch, scenes will be colored according to information stored in the client profile. In this way, someone who frequently goes winter–sporting, will be displayed slightly bronzed when he's looking for a white living–room furnishing.
- The **market profile** contains the supplier's marketing vision: what does he want to sell and on which way does he want to profile himself. This strongly influences the presentation towards the customer. For a "middle of the road" store, for example, the presentation of weekly discounts will be of primary interest, while for a top–segment advisor careful looking at sales possibilities with the presentation of some beautiful, but almost unaffordable, products is more usual.

In general, information about the products and the environments in which the products are to be used, has to be become available first. The first kind of information can be provided by a branch–organization; the second kind will have to come from some other, external, service provider. With this information, a general, person–independent view at the product can be created. Adding supplier–specific marketing information will lead to a unique Point of Sales. When lastly a client profile is added to the scenario, the product and its setting are clearly recognized by the customer.

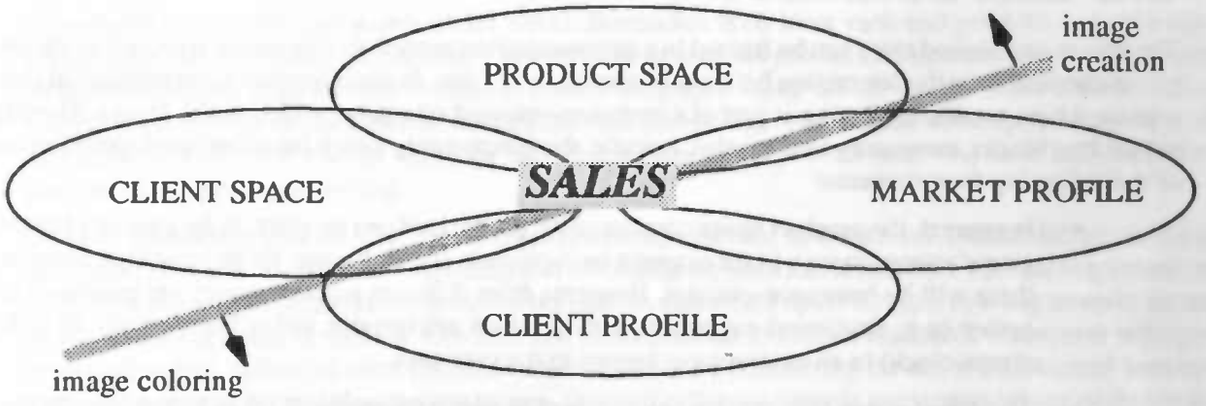



**Figure 1.3**: *Components of the communication model*

One can think of several scenarios. For demonstration purposes, for example, a fixed route will be followed, while for marketing purposes a fixed approach to a specific object suffices. Since a family usually consists of more than one person, there will be other family members moving around the room. Furthermore, it is imaginable that the user will walk around the room himself, changing direction as he pleases. In other words: there will be a certain interaction between MAGIC and the potential buyer, in which an intelligent dialog is necessary, possibly involving natural language. A continuous interpretation of the movements in the scene is also an important part of the information feed–back. Because of this, it's senseless to base interaction on a series of frozen images (like in today's computer games).

In practice, product and client space have to be matched, in order to satisfy both customers and suppliers wishes as much as possible. When this is done, a product is found which the customer finds interesting and which the supplier wants to sell. This product then will be colored according to data from client space and client profile. From this emphasis on finding a compromise, the principal goal of MAGIC can be summarized as:

*"modelling non–verbal negotiation in a multimedia setting"*

### 1.1.3 The network

On the supplier side, less emphasis will be on dynamic frame generation and interactive use. Rather, the changing business plays a part, resulting in new software which has to be integrated into the existing structure: Business Re–Engineering. A division shows up in delivering product information on the one side, and acquiring specific information which can lead to a personal approach to the customer on the other side. Apparently there is bidirectional network traffic, and therefore it is recommendable to look closely at the various aspects of privacy protection.

When we want to offer the supplier the possibility to change his assortment or presentation in a flexible and inexpensive way, the use of a network is obvious: when we don't use a network, the supplier has to inform every user of every change he wishes to make. The need for user feed–back also favors the use of a network.

Which network, then, is suitable for the system? The use of modems is an inexpensive solution to the supplier, but this is not suitable because of the low speed and high costs for the users. The use of the Internet doesn't provide a solution either, because the users often use modems to connect to the Internet. Besides that, it is often very difficult to find information on the Internet.

More suitable candidates seem to be ISDN and the TV–cable network. The cable network is a suitable candidate, especially since the cable companies started implementing optic fibre networks. ISDN is fast, but it is also expensive and not in common use. The cable network can't be used as a data–network every-

where at the moment, but in the long term it will prove to be very suitable since a large number of people are already connected to it.

Besides users and suppliers, there are also providers on the network. They manage databases in which information about the user's environment or the user himself is stored. One can think about floor plans of houses and loan registrations through BKR (Bureau Krediet Registratie). The supplier will use these kinds of information for a personal approach towards the user. Insurance companies could present a scenario to potential customers in which they are involved in a car crash. In this situation, it will be their own car that gets damaged, and the crash will occur on a crossing which is known by the users to be a very dangerous one. This might persuade the customer to get a better car insurance.

*The product and client space will consist of databases. For this, existing systems are available, so this aspect needs little research. Combining product space and client space into an interactive 'walk' can be done with the aid of existing techniques (think of the engines of games like Doom, Heretic and Descent). This aspect may take some time to implement, but it will also take little research. An entirely new system has to be developed for finding a compromise between client profile and marketing profile. Research is needed into the way this compromise is found. This research consists of psychological and management aspects besides computing science aspects. Psychological aspects can be found in the research into the client profile, but also in designing the user interfaces.*

## 1.2 System components

Within MAGIC, several parties meet. On the supply side, we have suppliers, each with his own range of products, a marketing strategy to sell the products and a sales tactics to get to the eventual sale. On the demand side, we have users, each with his own needs, a life–style in which this need can express itself, and each living in a phase in which these needs will be of more or less importance. Between these parties, a natural hate/love relationship exists: when a product is not a necessity of life, selling it often is a question of turning suspicion into trust. For the process of sales, the moment of contact is needed first of all, but the potential buyer is shy of this contact, and prefers looking around undisturbed. With this, the challenge of marketing is outlined: informing the customer without forcing, but in such a way that the customer will get into contact, preferably while excluding any competition.

The primary goal of MAGIC is to provide the product–information in a game–like hi–tech environment whilst stimulating both personal orientation and brand reputation. At first, it seems like the middleman (a.k.a. intermediary, trader or agent) is not needed anymore, but after looking closely is shows that this is not the case. Rather, there will be a new arrangement of services by means of the Electronic Highway, the medium we have in mind. A first reason for keeping the middleman is the possibility to have a direct contact as a means to push sales. But earlier the middleman can be useful as a "data–integrator": databases from several suppliers can be integrated into a range of products that covers the needs of his clients.

*There are certain purchases that are directly related. This insight has led to the possibility to get a travel insurance at the travel agent. In this case, no extra effort is needed: there's just an extra pile of brochures on the counter. The situation of a house agent who is selling mortgages and life insurances besides houses is quite the same. A step further away is the post office, renting cars and selling office equipment. The motivation in this case is looking for an appropriate use of unused shop space, konsekwent doorgezet op basis van het kental "Omzet per vierkante meter". Waar verkoop–areaal geen factor is, kunnen ook andere redenen bestaan om produkten samen te voegen, zoals kompenserend gedrag. In de bovengeschetste situaties is vaak sprake van een eventueel veelvuldig, maar altijd kortstondig kontakt. Immers, de omzet groeit naarmate de tijd per verkoop afneemt (onder aanname van een volbezette verkoper). Bij MAGIC wordt deze zo kostbare tijd grotendeels bij en door de klant besteed (gratis dus); daarnaast is een konstante monitoring van de klant mogelijk waardoor een betere en gerichtere begeleiding kan plaatsvinden.*

Bij MAGIC is er een simpele, technische reden om zoiets als een tussenpersoon te hebben: wanneer alle leveranciers en alle gebruikers in direct kontakt kunnen staan, ontstaat een vrijheid van keuze die eerder

verwarrend en daarom contra–produktief werkt. Bovendien is de klant vaak niet produkt–, maar koop–georiënteerd. Er is vermoedelijk behoefte aan een tussenpartij, die produkten samenstelt tot aanbiedingen en die individuele, vage wensen verzamelt tot een duidelijke vraag–uit–de–markt. Deze situatie is dus vergelijkbaar met die van de kleine kruidenier die voor Douwe Egberts de koffie verkoopt, met dit verschil dat onze kruidenier niet zomaar verkoopt maar een service toevoegt zoals deur–aan–deur sales in de bejaardenflat of avondopenstelling.

De tussenpersoon is voor MAGIC dus een gelijkwaardige partner, de Dritte–im–Bunde. Hij stemt vraag en aanbod op elkaar af, relateert marketing aan levensstijl en sales aan levensfase. Met andere woorden, hij is de hoekman van de markt die zowel op het palet in aanbod als op de variëteit in vraag inwerkt vanuit de bestaande regionale diversiteit. Om zulk een funktionaliteit te kunnen realiseren (en vooral wijzigen en onderhouden) zullen we eerst trachten vanuit een verdere detaillering tot een architektureel koncept met grote mate van herbruikbaarheid te komen. Daarom zullen we eerst de partijen op het net en de netdelen nader beschouwen.

### 1.2.1 Leverancier

De leverancier dient reeds een volwassen niveau van automatisering te hebben bereikt waarbij alle produkt–informatie voor voorraadbeheer en werkplanning in een centrale database is opgeslagen. In veel gevallen zal deze produkt–database een direkte relatie aangeven met een soortgelijke database voor fabrikage–gegevens binnen de fabrikage–afdeling. Op deze wijze hoort "product–on–demand" tot de direkte mogelijkheden.

De leverancier is in het bezit van een database met produkten in een gestandaardiseerd formaat (b.v. AUTOCAD), een verkoopprioriteit voor het produkt (dit geeft aan hoe graag de leverancier dit produkt wil verkopen) en een beschrijving van het produkt (zoals naam, type–informatie en prijs). Verder kan de leverancier de manier van verkopen richting geven (zoals bijvoorbeeld agressief of rustig). Deze gegevens zendt de leverancier op verzoek naar de gebruiker of stelt deze via een derde, een extra Provider, ter beschikking.

In de MAGIC–architektuur, zoals deze aan de leverancier–zijde noodzakelijk is, zien we de primaire processen van Figure 1.2 weer terug. De produktruimte wordt gegeven door de delen die uit de produktie–lijn komen, waarbij nog differentiatie op basis van klantenvoorkeur kan optreden. Deze delen kunnen onderdelen van het feitelijke produkt zijn; in die gevallen zal marketing tot een specifieke samenstelling besluiten. Deze marktprodukten worden door verkoop aan de markt gepresenteerd met aanwijzingen over beschikbaarheid, levertijd en speciale kortingen. De reaktie van de markt in de vorm van koopkontrakten worden via Verkoop ingenomen en via Inkoop en produktie intern verder verwerkt. Al deze interne gegevensstromen dienen natuurlijk op elkaar afgestemd te worden.

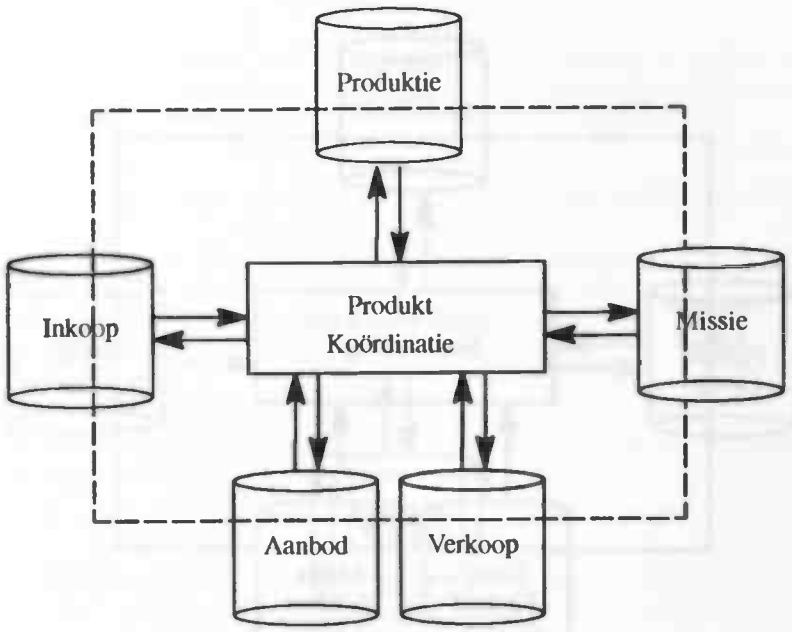



**Figure 1.4**: *Architektuur bij de leverancier*

Een overweging van groot belang is bij deze architektuur de onderlinge afstemming en interne organisatie van de processen Marketing en Sales. Binnen de discipline Bedrijfskunde is dit aan nogal grote bewegingen onderhevig. In de MAGIC–architektuur doen we daar vooralsnog geen andere uitspraak over dan dat eventuele verschuivingen in de respectievelijke taakstellingen bij de synchronisatie van de processen weer in orde gemaakt worden. Daarmee bereiken we in eerste instantie een scheiding tussen de gegevens– en de menselijke organisatie.

Een aan deze problematiek gekoppelde vraag is die van Installatie en Onderhoud. Het MAGIC–koncept is gegrondvest op twee principes: (a) bestaande gegevensbestanden zijn toegankelijk over reeds bestaande programmatuur, en (b) koppelingen tussen deze bestanden geschiedt middels relaties, die vanuit de praktijk (lees voorbeelden) leerbaar zijn. Door verder een grote mate van software hergebruik te handhaven zal de architektuur niet alleen transparant maar ook overdraagbaar naar andere platformen zijn.

### 1.2.2 Tussenhandel

De tussenhandelaar fungeert als bemiddelaar tussen de leverancier en de klant. Zijn voornaamste bezigheid is het verpakken van de diverse produkten tot een variëteit aan aanbiedingen zoals die in de primaire interesse van groepen klanten liggen. Uit dien hoofde zal hij naar beide zijden kontakt zoeken: van de leveranciers ontneemt hij de produktbeschrijvingen en aan de klanten levert hij "koopjes". In feite is de tussenhandelaar de veralgemenisering van de "kruidenier om de hoek", maar dan binnen de funktionaliteit die op de Elektronische Snelweg ter beschikking is.

De belangrijkste bezigheid van de tussenpersoon bestaat in de differentiatie van het marktaanbod naar produktgroepen en gebruikersgroepen. Dit alles onder het motto dat "de klant koning is" en dat hij dus niet met overbodige zaken lastig gevallen dient te worden. In zijn eigen belang zal hij deze differentiaties gaan inkleuren op basis van verkoopmarges en soortgelijke afname–verplichtingen. Verder zal hij zich omzetshalve bezig houden met de samenstellen van standaard–demo's. We zullen ook hier weer een architektuur met vier gezichten tegenkomen.

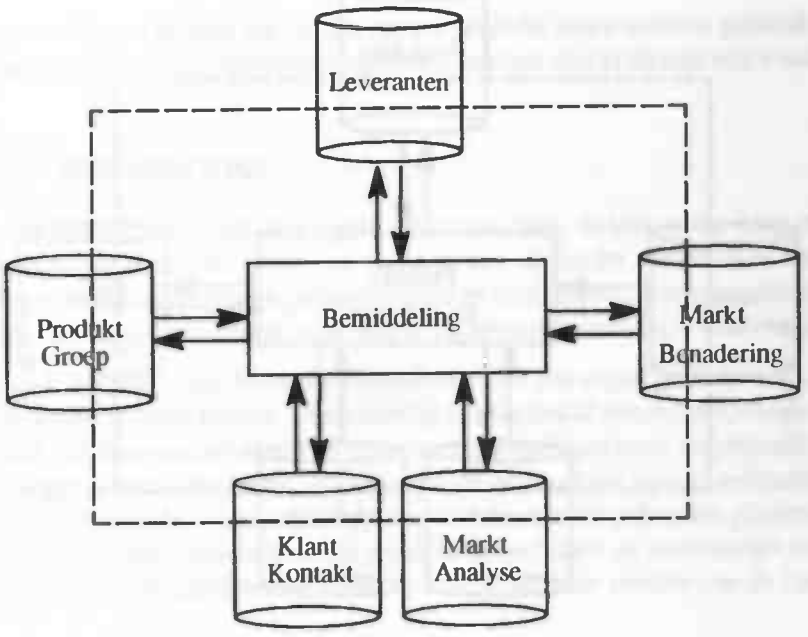



**Figure 1.5:** *Architektuur bij de tussenpersoon*

*Als voorbeeld geven we hier kort de situatie weer rond het kopen van een tweedehands–auto. Daarbij komen als "leveranciers" in aanmerking: de verschillende auto–importeurs samen met hun lokale gelicenseerde automobielbedrijven voor de gangbare prijzen, de ANWB voor een tweede oordeel en het jaarlijkse onderzoek naar de onderhoudskosten en de verzekeraars voor de mogelijke schade–kosten en de poliskosten. De klant zal bij het kopen van een tweedehands–auto immers mede geïnteresseerd zijn in de te verwachten bijkomende kosten teneinde een volledig en toch op zijn persoonlijke financiële en sociale situatie toegesneden advies als uitgangspunt voor zijn beslissing te kunnen nemen. Voor het garage–bedrijf zal dit geen grote gevolgen hebben, maar voor het verzekeringsbedrijf is dit een mogelijkheid zich met een persoonlijk advies al direct tijdens de aankoop bekend te maken.*

Uit het bovenstaande voorbeeld blijkt al direct, dat de ingewonnen gegevens niet 1–1–duidig hoeven te zijn. Een vorm van arbitrage of een zekere margebepaling is noodzakelijk, maar dit leidt tevens tot een vorm van toegevoegde waarde: de klant wordt immers op de variabiliteit van de kosten gewezen. We moeten dus hier al beginnen om te gaan met vaagheid en deze behouden zonder door middelingen of statistiek op quasi–exakte feiten over te stappen.

### 1.2.3 Gebruiker

De gebruiker is natuurlijk het uiteindelijke doelwit van de MAGIC architektuur. Zonder de klant is er geen verkoop en het feitelijke doel is om deze persoon een zo eerlijk maar tevens zo suggestief mogelijk beeld van zijn komende aankoop en de konsekwenties op zijn levenssituatie te geven. De gegevens moeten door hem niet zozeer gelezen als wel beleefd worden. Met andere woorden, de beoogde gegevensstromen dienen in een persoonsgerichte visualisatie uit te monden.

Ook bij de gebruiker komen we weer de vier gezichten van de MAGIC architektuur tegen. Van de leveranciers en/of de tussenpersonen komen produktgegevens ter beschikking, maar om deze persoonsgericht te kunnen vinden en gebruiken zal tevens een vaststelling van de persoonskarakteristiek van de klant moeten plaatsvinden. In de toonzaal doet de verkoper dit intuïtief: een goede verkoper "voelt aan" wat de klant wil en hoe deze benaderd moet worden. Verder moet de levenssituatie van de klant ingeschat worden en daarmee samenhangend een profiel van wat de verkopende organisatie het liefst kwijt wil. We komen daarmee op de in zijn grondvorm al eerder getoonde weergave van de lokale architektuur in Figure 1.6.

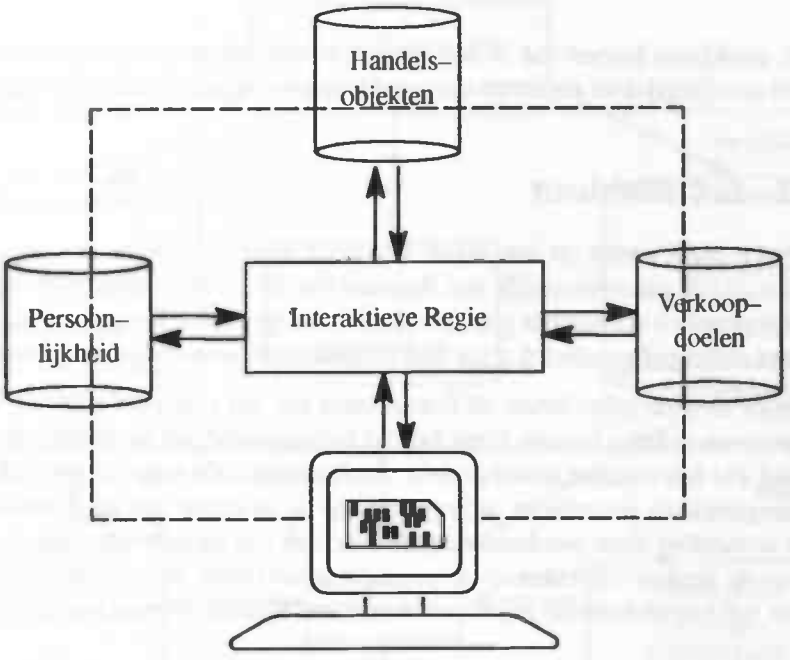



**Figure 1.6**: *Architektuur bij de gebruiker*

*Voor een toelichting op de funktionaliteit bij de gebruiker noemen we hier de keuze van huiskamer–meubilair: een situatie die in de gangbare praktijk vaak leidt tot veel verkeer op straten en parkeerterreinen voor het herhaald bezoeken van grote meubelshows. Niet voor iedereen is dit een onverdeeld genoegen en MAGIC geeft hier de mogelijkheid om vanuit kantoor en/of huis al een gedegen, persoonsgerichte voorselektie te maken, te beleven en te delen met de partner.*

Vanuit de eerder genoemde visie op de ontwikkeling van de hardware, gaan we uit van krachtige, maar goedkope, multimedia–PCs met veel opslagruimte. Dit biedt als voordeel dat er lokaal (bij de gebruiker op de PC) veel kan worden verwerkt, zodat het netwerkverkeer zoveel mogelijk beperkt kan worden. Dit brengt de kosten omlaag en verhoogt de snelheid.

Alle persoonlijke informatie (klantprofiel) kan lokaal worden opgeslagen, liefst op een verplaatsbaar opslagmedium zoals een floppy disk of een smart–card. Voordeel hiervan is dat deze informatie meegenomen kan worden naar een willekeurige computer, waardoor de gebruiker altijd op een persoonlijke manier benaderd kan worden door de leverancier. Verder voorziet het in de behoefte tot beveiliging van persoonlijke gegevens, omdat bij diefstal van hardware niet noodzakelijkerwijs ook de kritische persoonlijke elementen ontvreemd zullen worden.

## 1.3 Het gebruikssysteem

In het voorgaande hebben we de gegevensstromen binnen MAGIC al enigszins aangegeven. Een verdere detaillering moet nog plaatsvinden, maar we kunnen al wel een voorbeeld–systeem gaan invullen. Hiermee verkrijgen we een voorbeeld, waarmee de andere MAGIC–delen hun voordeel kunnen doen. We kiezen hiervoor het K3–deel: de funktionaliteit binnen MAGIC die bij iedere klant geplaatst kan worden. Zoals zal blijken komen de diverse innovatieve gegevensbewerkende delen hierin al voor, zodat het bouwen van K3 al een goed zicht op de te behalen funktionaliteit zal bieden. Bovendien liggen aan de gebruikerskant de grootste problemen, wat betreft de real–time bewerking van databases.

Onafhankelijk van de organisatie van de gegevensloop over het Netwerk zal de klant c.q. gebruiker kunnen uitgaan van de lokale beschikbaarheid van de leveranciergegevens. Binnen het K3–programma zien we dit terug als de **applikatielaag**. De (indirecte) interaktie tussen leverancier en klant levert mogelijk wijzigingen in marktprofiel en klantprofiel op. Een leverancier kan bijvoorbeeld aan de hand van wensen

van klanten tot de konklusie komen dat er een nieuw produkt moet worden gemaakt. Het klantprofiel wordt bijvoorbeeld gewijzigd door akties en keuzes die de klant maakt tijdens een sessie met een leverancier.

### 1.3.1 De K3–architektuur

Het K3–programma bestaat verder uit drie lagen: de sektor–laag, de konversie–laag en de exekutie–laag (Figure 1.7). Hierin wordt successievelijk aan de hand van de eerder verzamelde karakterisatie van de klant een scenario gepland en vervolgens gekonverteerd en uitgevoerd. Tevens wordt een interpretatie van de interaktieve (niet altijd geformuleerd maar wel in daadvorm aanwezige) klantwensen teruggemeld.

*Het programma voor de gebruiker bevat de bouwstenen om het eigen gezicht van de leverancier (het marktprofiel) naar voren te laten komen. Denk hierbij bijvoorbeeld aan mogelijkheden voor het afspelen van beeld en geluid. De leverancier stuurt in feite slechts insteldata naar de gebruiker. Het programma wordt daarmee aangepast en vervolgens 'afgespeeld' op de machine van de gebruiker. Zo kan bijvoorbeeld een virtuele wandeling door een keuken op de machine van de gebruiker plaatsvinden als eenmaal de beschrijving van de keuken (objekten uit de produktruimte) door de leverancier naar de gebruiker is gestuurd. Er zullen voor verschillende marktsektoren verschillende versies van de basisprogrammatuur bestaan.*

De **sektor–laag** bevat alle algemene routines die voor een bepaalde gebruikssektor (bijvoorbeeld binnenhuisarchitektuur) nodig zijn. Vanuit de applikatie–laag wordt de applikatie–specifieke informatie aan de sektor–laag doorgegeven. Op deze manier is een programma ontstaan dat bijvoorbeeld de keuken–verkoop van Bruynzeel implementeert. De volgende komponenten zijn in deze laag te vinden:

- de Intelligente PersoonsIdentifikatie (IPI); hierin wordt aan de hand van een vragenlijst of een standaard demonstratie eenmalig een profiel van de klant opgebouwd in termen van leeftijd, salariskategorie, tendens tot luxueuze wensen etc..
- de Intelligente Strategie Beslisser (ISB); hierin worden stelselmatig de wensen van de klant in overeenstemming gebracht met mogelijke scenario's zoals verkrijgbaar van de leverancier of tussenpersoon ten einde tot de te volgen verkoopstrategie te geraken.
- de Intelligente GedragsIdentifikatie (IGI); hierin worden de gevolgde scenario's en de reakties daarop van de klant geïnterpreteerd ten einde tot geschikte vernieuwingen aan de aanbodzijde te geraken.
- de VerkoopMachine (VM); in deze komponent vindt de uiteindelijke Sales plaats. De verkoper bepaalt aan de hand van de instellingen van de leverancier, de produkten, verkooppriorïteiten en de verkoopstrategie zoals boven besproken het te gebruiken verkooppraatje. Wijzigingen in de verkoopstrategie tijdens het verkoop–praatje worden hier ook opgevangen. De verkoper vormt samen met de instellingen uit de applikatie–laag het marketingprofiel.

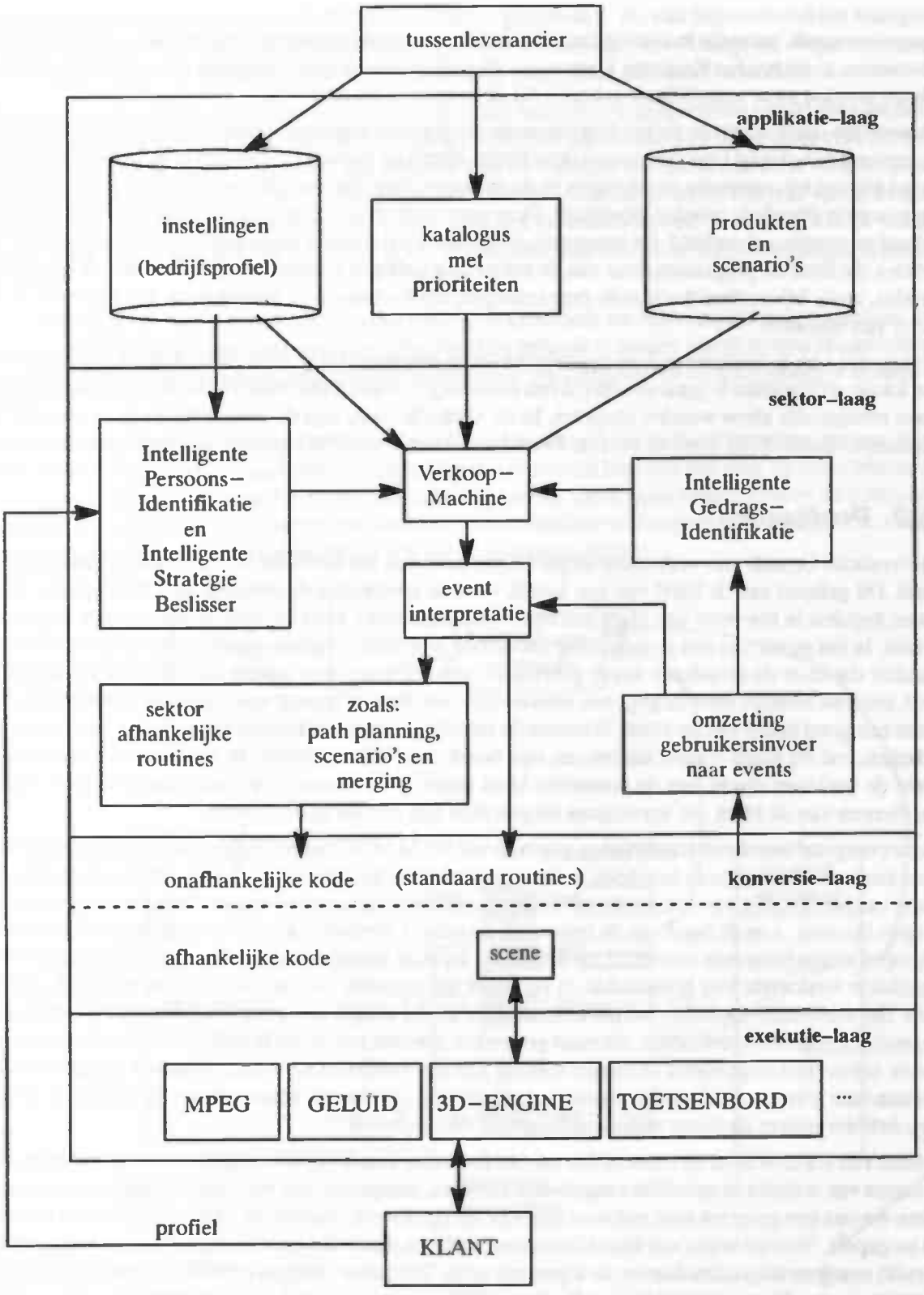



**Figure 1.7**: *De lokale architektuur*

De aankleding van de scenario's vindt zijn oorsprong in twee bijdragen: (a) de vast ingebakken uitvoering voor een specifieke gebruikssektor binnen de sektor–afhankelijke routines, en (b) een verdere invulling vanuit de applikatie–laag. Hierbij dient bedacht te worden, dat weliswaar slechts één tussenleverancier

aangegeven wordt, maar dat in zijn algemeenheid hier ook gegevens van andere data–leveranciers worden ingewonnen, zoals b.v. het Kadaster. Deze verschillende soorten gegevens hebben een verschillende betekenis en worden vaak verschillend gebruikt bij de konversie tot animatie.

De **konversie–laag** maakt de verbinding tussen de programma's in de exekutie–laag en sektor–laag. Deze laag verzorgt een stabiel interface tussen deze beide. Hierdoor kunnen de routines in de sektor–laag ongewijzigd blijven bij eventuele wijzigingen in de exekutie–laag. Slechts de routines in de konversie–laag dienen in zo'n situatie te worden gewijzigd. Deze laag biedt aldus een hoge graad van software herbruikbaarheid en overdraagbaarheid. De konversie–laag voorziet dus onder andere in een aantal standaard 3D–operaties die door de programmatuur van de sektorlaag gebruikt kunnen worden. Het betreft hier basis–operaties, zoals bijvoorbeeld afstands–berekeningen, het toevoegen en verwijderen van objekten en het roteren van objekten.

De integratie van de onderscheidene gegevensstromen geschiedt bij de gebruiker in een K3–programma. Deze lokale architektuur is gerangschikt in een aantal lagen, waarin successievelijk de algemene gegevens tot een multimedia show worden omgezet. In de **exekutie–laag** zijn de programma's te vinden die het uiteindelijke multimedia werk uitvoeren. Denk hierbij aan een MPEG–player, 3D–engine, MIDI–player etc..

## 1.3.2 Profilering

De leverancier bepaalt zijn verkoopstrategie aan de hand van het beeld dat hij zich van de klant heeft gevormd. Dit gebeurt aan de hand van een aantal, voor de leverancier dus belangrijke, kenmerken. Om te kunnen bepalen in hoeverre een klant aan een bepaald kenmerk voldoet, stelt de leverancier vragen aan de klant. In het geval van een grootschalig onderzoek zijn deze vragen nog genoteerd op een standaard–formulier dat door de enquêteur wordt gebruikt. Omdat de enquêteur samen met de klant het formulier invult, ontstaat behalve door de gegeven antwoorden ook door de manier waarop de antwoorden tot stand komen een goed beeld van de klant. Wanneer de enquêteur bepaalde kenmerken nog niet voldoende kan inschatten, zal hij extra vragen stellen om zijn beeld compleet te maken. In het kleinschalige kontakt, waarin de verkoper direkt met de potentiële klant praat, is de beeldvorming niet fundamenteel anders: klassificeren van de klant om vervolgens tot een plan van aanpak te beslissen.

In deze paragraaf wordt een beschrijving gegeven van het MAGIC bestanddeel voor intelligent klassificeren en beslissen. Dit systeem begeleidt de leverancier bij het bepalen van de juiste verkoopstrategie. Met behulp van het invullen van een standaard–formulier probeert het systeem een profielschets van de gebruiker op te bouwen. Aan de hand van dit ingevulde standaard–formulier kan de leverancier besluiten welke verkoopstrategie het meest succesvol zal uitpakken bij deze specifieke klant. Hiermee wordt de al eerder aangeduide werkwijze van leverancier en verkoper ten opzichte van de klant zoveel mogelijk gehandhaafd. Het standaard–formulier zal per kenmerk een aantal vragen bevatten. De beoordeling van de door de gebruiker gegeven antwoorden, normaal gesproken een taak van de verkoper, wordt in het systeem door neurale netwerken uitgevoerd, in samenwerking met een beslissingssysteem. Wanneer de gegeven antwoorden naar tevredenheid van het systeem zijn ingevuld (voldoende betrouwbaar zijn), dan kan met een fuzzy kennissysteem de juiste verkoopstrategie worden bepaald.

In plaats van via de vragen en antwoorden van het formulier kan de informatie ook verkregen worden door het meten van reakties in specifiek aangeboden situaties. Aangezien een veelvoud mogelijkheden bestaat stellen we ons een generiek stuk software ten doel dat op diverse plaatsen en in diverse situaties kan worden toegepast. Voor de wijze van klassificeren en beslissen maakt het geen verschil welk invoer–medium gebruikt wordt en we zullen daarom de algemene term "formulier" blijven gebruiken. Voor de klassifikatie stellen we een Connectionist Expert Systeem (CES) voor: een lerend netwerk dat onder omstandigheden als Fuzzy software gerealiseerd kan worden; voor de beslissing kunnen we dan volstaan met een klein conventioneel expert systeem dat vanuit de CES de gegevens altijd in een toepassingsonafhankelijk formaat toegeleverd krijgt.

Het **standaard–formulier** bevat per kenmerk één of meer vragen. Omdat de antwoorden op deze vragen als invoer dienen voor een neuraal netwerk, moeten deze antwoorden uiteindelijk numeriek zijn. De invul-

ling van het formulier kan op verschillende manieren geschieden. Zo kan bijvoorbeeld het klantprofiel, of gegevens van de leverancier over deze klant gebruikt worden om zoveel mogelijk vragen automatisch te beantwoorden. Het systeem hoeft de gebruiker deze vragen dus niet meer te stellen.

Wanneer het standaard–formulier ingevuld is, worden de antwoorden aangeboden aan de **neurale netwerklaag**. Deze laag kent per kenmerk een neuraal netwerk. Elk input–neuron van zo'n netwerk correspondeert met een vraag van een kenmerk. Het netwerk kent twee output neuronen, een maat voor het kenmerk en een betrouwbaarheidswaarde van deze maat. Deze betrouwbaarheidswaarde hangt af van de konsistentie tussen de antwoorden op de vragen over een kenmerk en eventueel van de zwaarte van de vraag. De kenmerken zijn zo gekozen dat aan de hand van deze kenmerken een juiste verkoopstrategie bepaald kan worden.

De kenmerken, die uit de neurale netwerken komen, worden door het **kennissysteem** gebruikt om tot een verkoopstrategie te komen. Dit kennissysteem zou, door gebruik te maken van de betrouwbaarheidswaarden van de kenmerken, ook een betrouwbaarheidswaarde voor de gekozen verkoopstrategie kunnen opleveren.

Het **beslissingssysteem** controleert of de uitvoer van het neurale netwerk betrouwbaar genoeg is om te gebruiken. Als de betrouwbaarheidswaarde te laag is, zal worden besloten om over de minst betrouwbare kenmerken extra vragen te stellen uit een centrale vragenpool. Deze extra vragen zouden de betrouwbaarheid moeten vergroten. Ook wanneer het kennissysteem meerdere verkoopstrategieën oplevert, zullen extra vragen gesteld moeten worden om tot een keuze te kunnen komen.

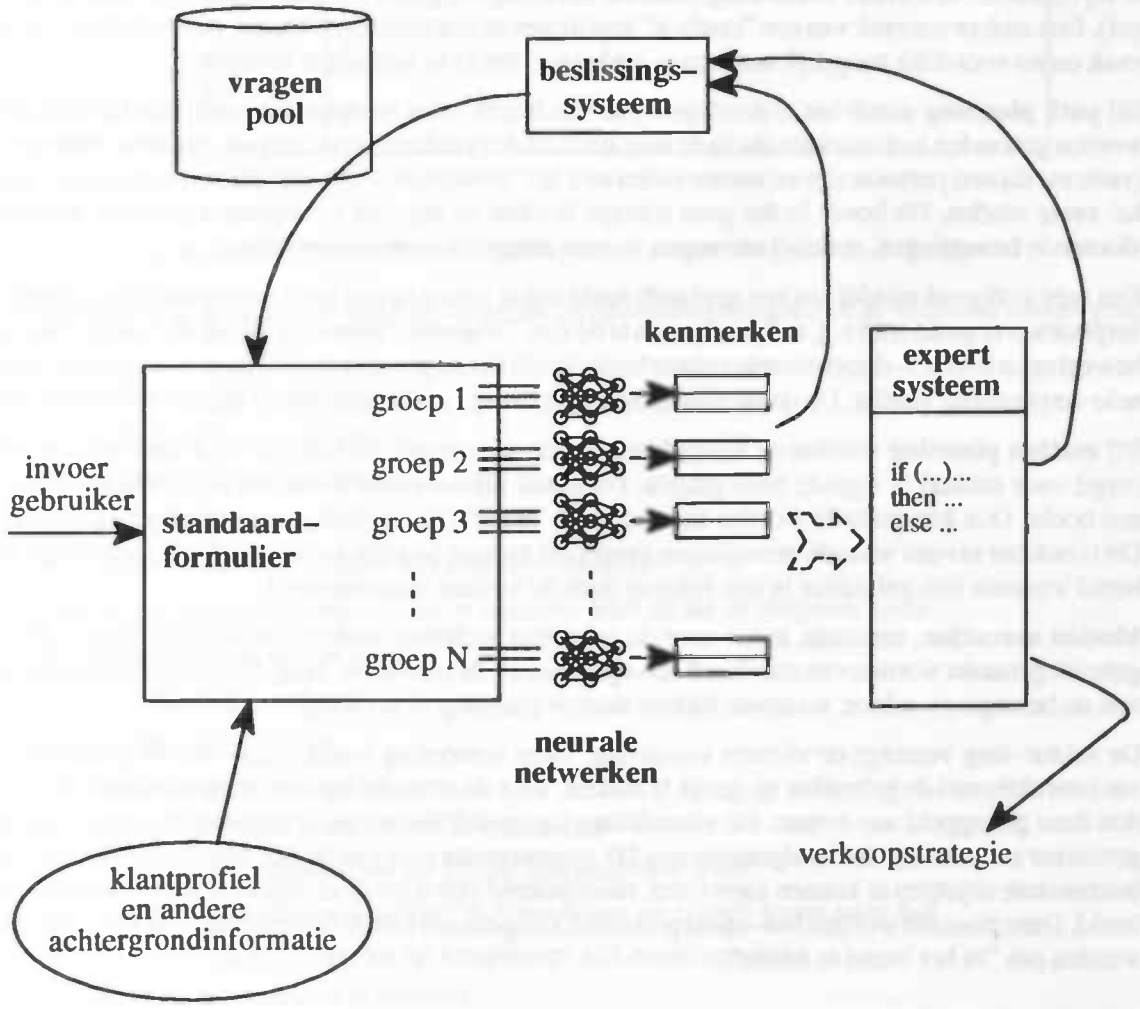



**Figure 1.8**: *De profileringsarchitektuur*

*Wanneer een gebruiker tot een koop besluit, vormt zijn gebruikshistorie samen met het opgebouwde profiel een deel van de trainset die bij volgende klanten gebruikt kan worden. Ook het gebruik voor een betere service verlening door de leverancier kan niet uitgesloten worden. Deze historie heeft dus een specifieke waarde en zou een reden kunnen zijn voor een separate Service Provider op het Net. De trainset is onlosmakelijk verbonden aan de applikatie–programmatuur. Vanuit de leverancier en/of de tussenpersoon zal bij de voorziene generieke software dus slechts deze trainset uitgewisseld moeten worden om een nieuwe functionaliteit van het systeemdeel mogelijk te maken.*

### 1.3.3 Animatie

In de sektor–laag wordt tevens een virtuele wereld gedefinieerd: de scene. Door middel van aanroepen van routines uit de konversie–laag wordt deze virtuele wereld uiteindelijk opgeslagen in een voor de 3D–engine (uit de exekutie–laag) begrijpelijk formaat. De scene bestaat uit objekten, die weer bestaan uit één of meerdere basisvormen zoals boxen, cilinders en vlakken. Verder staat in de scene beschreven waar elk objekt zich bevindt. Tenslotte kunnen er akties worden gedefinieerd die aangeven wat er moet gebeuren wanneer het betreffende objekt wordt geactiveerd, bijvoorbeeld door op een muistoets te klikken. De bewegingen in de virtuele wereld kunnen worden uitgesplitst in vier onderdelen: (a) target planning, (b) path planning, (c) motion planning, en (d) motion execution.

Bij **target planning** wordt bepaald wat het begin– en eindpunt is van de uit te voeren beweging. Wanneer er bijvoorbeeld een kraan wordt aangeklikt zal het eindpunt gelijk zijn aan een positie voor de kraan (target). Een andere oorzaak van een "doelwit" kan liggen in een verkoopstrategie, die bedoeld is om de klant vaak en zo voordelig mogelijk met een te verkopen objekt in kontakt te brengen.

Bij **path planning** wordt het te doorlopen pad van begin– naar eindpunt bepaald. Hierbij moet rekening worden gehouden met objekten die in de weg staan en de fysieke eigenschappen van het te bewegen objekt (vaak zal dit een persoon zijn en samenvallen met het 'viewpoint'). De path planning moet een 'menselijke' route vinden. Dit houdt in dat geen scherpe hoeken en zig–zag bewegingen gemaakt worden, maar vloeiende bewegingen, waarbij omwegen zo veel mogelijk vermeden worden.

Een zeer geëigend middel om een geplande route om te zetten in een serie verplaatsingen alsmede om de verplaatsen te genereren c.q. te konserveren is de zgn. "trajectory planning" uit de Robotica. Daar worden bewegingen in een 3–dimensionele ruimte bestudeerd; in ons geval is slechts de eenvoudiger 2–dimensionele verplaatsing vereist. De derde dimensie wordt "vaag" eraan gekoppeld binnen de motion planning.

Bij **motion planning** worden de bewegingen frame voor frame uitgewerkt. Ook hier moet worden gezorgd voor natuurlijk ogende bewegingen. Dit houdt bijvoorbeeld in dat het hele lichaam meedraait in een bocht. Ook kan gedacht worden aan een soort 'hupje' (de Heretic–loopbeweging) tijdens het lopen. Dit is ook het niveau waar de bewegingen aangepast kunnen worden aan de specifieke gebruiker, bijvoorbeeld wanneer een gebruiker in een rolstoel door de keuken manoeuvreert.

**Motion execution**, tenslotte, is het voor de gebruiker zichtbaar maken van de bewegingen. Hierbij zal gebruik gemaakt worden van standaard 3D–operaties uit de konversie–laag. Dit deel maakt tevens gebruik van de bewegings–editor, waarmee tijdens motion planning al ervaring is opgedaan.

De sektor–laag verzorgt de virtuele wandeling. Deze wandeling wordt event–driven geïmplementeerd, om interaktie met de gebruiker mogelijk te maken. Voor de afhandeling van de input–mogelijkheden worden deze gekoppeld aan events. De afhandeling van muisklik–events vraagt een iets andere aanpak. De gebruiker zal namelijk in het algemeen een 2D–representatie zien van de 3D–wereld. Om nu met een muis interessante objekten te kunnen aanwijzen, moet bekend zijn waar deze objekten zich bevinden op het 2D beeld. Deze plaatsen worden **hot–spots** genoemd. Omgekeerd kan de wetenschap over hot–spots gebruikt worden om "in het beeld te meten".

## 1.4 Voorbeeld

*Als voorbeeld behandelen we hier de keuken–verkoop applikatie. Een keukenleverancier zal de gebruiker een keuken presenteren, waar de gebruiker doorheen geloodst wordt of zelf doorheen kan lopen. Voor deze uitwerking wordt de keuken gebruikt uit Figure 1.9. Een gebruiker doet twee stappen naar voren en draait iets naar rechts. Op dat moment krijgt hij een keukenkraan in zicht, die hij vervolgens met behulp van de muis aanklikt. De gebruiker zal dan op een natuurlijke wijze naar de kraan lopen en deze van spoelbak B naar A zwenken. Na het aanklikken van de kraanknop zal de gebruiker de kraan laten stromen.*

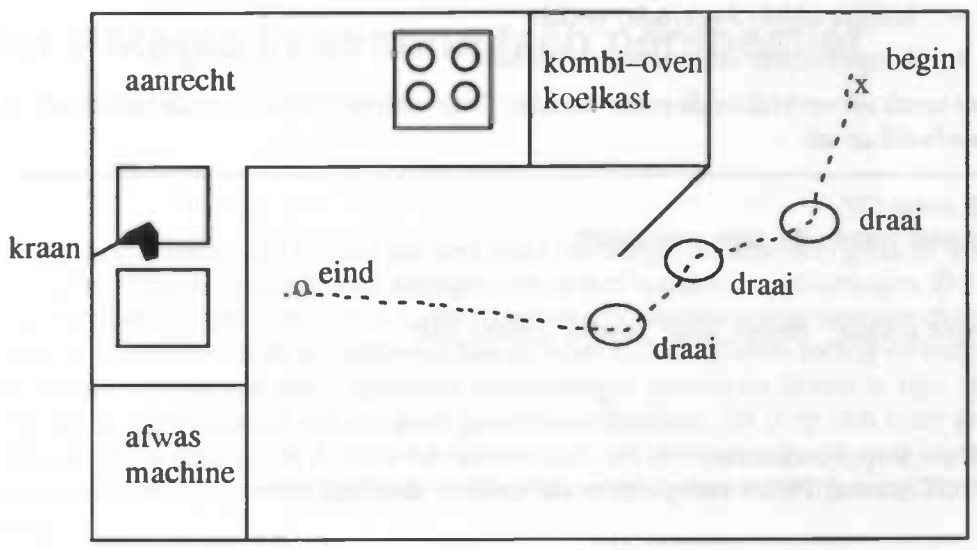



**Figure 1.9**: *Een persoon loopt vanuit positie x naar de kraan in de keuken*

*Voor dit systeem zijn in de sektor–laag de volgende input–mogelijkheden aan de volgende events gekoppeld:*

- *pijltje omhoog: naar voren.*
- *pijltje omlaag: naar achter.*
- *pijltje rechts: draai rechts.*
- *pijltje links: draai links.*
- *linker muisknop: hotspot van objekt aangeklikt.*

*Voor het uitvoeren van het bovenbeschreven scenario leidt dit tot de volgende serie:*

- *[Opm.] kraan niet in zicht*
- *[Aktie] pijltje omhoog*
- *[Aktie] pijltje rechts*
- *[Aktie] pijltje omhoog*
- *[Opm.] kraan in zicht, achter koelkast deur*
- *[Aktie] muis positioneren op kraan, klik muisknop links*
- *[Opm.] loopt om koelkastdeur richting kraan en zwenkt kraan naar bak*
- *[Aktie] muis positioneren op kraanknop, klik muisknop links.*
- *[Opm.] kraan begint te stromen*

*Deze input akties hebben de volgende events tot gevolg:*

- *naar voren*
- *draai rechts*
- *naar voren*
- *draai rechts*
- *naar voren*
- *draai rechts*
- *naar voren*
- *hotspot objekt 34 (kraan) geklikt*
- *hotspot objekt 40 (kraanknop) geklikt*

*Elk event wordt verwerkt door de event–handler. Deze activeert vervolgens de bijhorende routine. Dit ziet er bijvoorbeeld zo uit:*

```
CASE event OF
  stap naar voren:  do_stap_voorwaarts;
        :
        :
  hotspot geklikt:   object_actie_handler [object_id];
        :
END;

PROC do_stap_voorwaarts;
  IF NOT botsing THEN viewpoint := viewpoint + richtingsvector;

PROC object_actie_handler (object_id);
  CASE object_id OF
    :
    34 : target = 'vlak bij kraan, tegen aanrecht'; /* target planning */
         pad = pad_planning (huidige_positie, target);
         gekleurde_loopbeweging (pad);
         /* gekleurde loopbeweging is een bibliotheekroutine, die het Heretic–loopje implementeert,
            rekening houdende met bijvoorbeeld de werkelijke lengte van de gebruiker. In deze
            routine vindt dus de motion planning en execution plaats.*/
```

*Na aanklikken van hotspot objekt 34 wordt eerst bepaald waar naartoe de beweging moet geschieden (target). Hierna moet een pad worden gevonden tussen het begin– en eindpunt. In dit geval moet de koelkastdeur worden omzeild. Het omzeilen moet op een natuurlijke manier geschieden, zoals in Figure 1.9 te zien is. Wanneer de target bereikt is, zal de aktie* ***do–zwenk–kraan*** *worden uitgevoerd, die er bijvoorbeeld als volgt uit ziet:*

```
PROC do_zwenk_kraan;
    beweeg_hand_naar_kraan;
    zwenk_kraan (A);
```

*Deze twee procedures maken gebruik van operaties uit de konversie–laag en uit de sektor–laag. Met de routines uit de sektor–laag wordt gepoogd de gebruiker op een persoonlijke manier te benaderen, bijvoorbeeld een hand met gelakte nagels, een zwarte hand etc.. Voor beide akties is geen target planning, path planning etc. nodig, omdat het hier gaat om duidelijk gespecificeerde akties. Wel moet gecontroleerd worden of er collisions ontstaan door het uitvoeren van de akties.*

# Chapter 2 Magic in ekonomisch perspectief

Bij de term klant moet binnen MAGIC aan een heel scala van mogelijke afnemers gedacht worden: van producent met grootschalige faciliteiten tot kleingebruikers met openbare voorzieningen. Het ligt in de lijn der verwachting dat in eerste instantie de kapitaalkrachtige producenten aangesproken dienen te worden. Pas in een later stadium zal de tussenhandel aan de beurt kunnen komen, terwijl de burgerman als laatste komt omdat daarvoor een aantal openbare voorzieningen geschapen dienen te zijn. Men denke daarbij vooral aan de aanwezigheid van een goed geoutilleerd kabelnet. Dit is op zich zeker geen utopie en het is denkbaar dat dit uiteindelijk de meer lukratieve markt zal blijken te zijn. Voor de exploitatie van deze openbare markt met zijn vanzelfsprekende service–behoefte is echter mogelijkerwijs een andere organisatie nodig.

Algemeen wordt verondersteld dat met de intrede van de Elektronische Snelweg tevens de tussenhandel zal verdwijnen. Dit lijkt in eerste instantie een terechte konklusie. *Disintermediatie* levert een direkte verbinding tussen eindgebruiker en producent en leidt door verwijdering van "het vet van de middelman" tot een kostenreduktie en een verbeterde mogelijkheid om direkt op de behoeften van de eindgebruiker te produceren. Beide lijken met deze deal te winnen.

Echter, bij een sterk geautomatiseerde massa–produktie is het niet interessant om op de wensen van de individuele klant in te gaan: een minimum volume van de vraag is immers noodzakelijk. Hoewel dit volume als som van individuele reacties zeker kan ontstaan, blijkt een makelende tussenpersoon steeds onmisbaar. Voorbeelden ziet men hedentendage b.v. in de telekommunikatie, waar tussen de "kabelexploïtant" en de telefoongebruiker een nieuwe tussen–business ontstaat. Een ander gezichtspunt is daarom, dat het ontstaan van een onafhankelijk opererende tussenhandel een teken is van de groei naar volwassenheid van een marktsegment. Er blijkt kennelijk steeds weer behoefte te bestaan aan bundeling van mogelijkheden zodat een gericht gebruik met minimale kennis mogelijk wordt. Wij citeren: *Reliable and well–supported packages are the best possible defense against the relentless price attrition.*

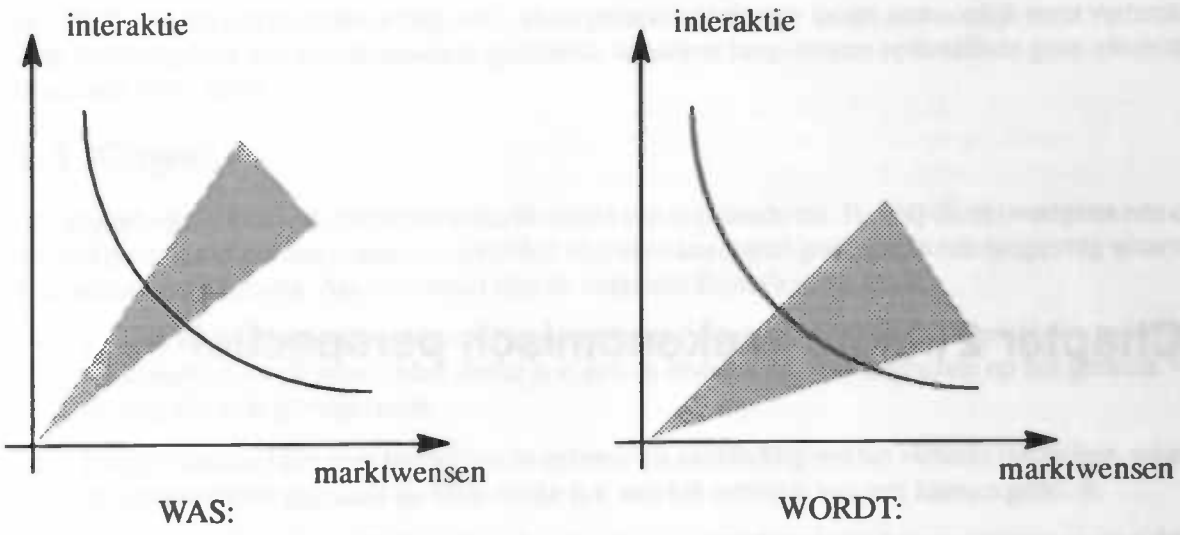



**Figure 2.1**: *Ontwikkeling van de tussenhandel in de markt*

In dit hoofdstuk zal eerst een verdere differentiatie gemaakt worden in de karakteristieke ontwikkelingsfasen van de diverse marktsegmenten. Daarmee wordt getracht een begin te maken met een poging de hoeveelheid inspanning te kwantificeren die nodig is om een bepaald marktsegment binnen de MAGIC filosofie op de Elektronische Snelweg te brengen. Vervolgens wordt vanuit een tweetal perspektieven een aantal marktsegmenten belicht en een verwachting uitgesproken hoe dicht deze deelmarkten reeds bij een ekonomisch zinvolle entrée op de Elektronische Snelweg staan. Op basis hiervan wordt besloten om in eerste instantie een algemeen raamwerk verder uit te werken teneinde de uitvoering van MAGIC in een prototype te visualiseren.

## 2.1 Werkgebied

Het MAGIC werkgebied is opgespannen langs drie assen of *views*: (a) de produkt–oriëntatie, (b) de interaktie–focus, en (c) de objekt–differentiatie. Langs ieder van deze drie assen zijn diverse oriëntaties aan te geven die in kombinatie met de plaatsbepaling langs de andere assen tot een unieke produkt–markt kans aanleiding geven. Met andere woorden, we gaan in het volgende uit van een 3–dimensionale wereld waarin de diverse aspekten van interaktieve visualisatie naar voren komen. Bij een verdere waardering van de haalbaarheid dient waarschijnlijk een aparte ijking plaats te vinden van de doelgroep: fabricant, tussenhandel of eindgebruiker. Hierbij is het vanzelfsprekend de veranderingen in de markt als geschetst in Figure 2.1 in herinnering te houden.

De bovenstaande verdeling is tevens in relatie met bestaande wetenschappelijke richtingen te interpreteren. Bij de produkt–oriëntatie zullen vooral elementen van de bedrijfskunde en de ekonomie terug komen; bij de interaktie–fokus zal vooral inbreng vanuit de psychologie en de sociologie aan de orde zijn, terwijl bij de objekt–differentiatie veelal de informatie–technologie (in bijzonder de informatika) een rol spelen. Dit tekent reeds de multi–disciplinariteit van de onderhavige problematiek. Wellicht ten overvloede zij er overigens op gewezen dat de hiernavolgende inschatting voornamelijk gebaseerd is op de bestaande situatie en derhalve de toekomst volledig verkeerd kan interpreteren, oftewel de analyse kan zeker geen markt–inzicht vervangen.

### 2.1.1 Produkt

Bij de **produkt–oriëntatie** is een aantal deelgebieden te onderscheiden die in hun nadruk lopen van de technologische (*technology push*) tot de marketing (*market pull*) mogelijkheden. Daarmee wordt in feite

het spanningsveld tussen marktvraag enerzijds en techniekdruk anderzijds in kaart gebracht, waarbij gekwantificeerd kan worden in hoeverre de beoogde ontwikkeling aansluit op bestaande maatschappelijke en/of technologische processen. Als voorbeeld van zo'n verdeling is te noemen:

- *[definitie]* de funktionaliteit van elke MAGIC uitvoering zal gedefiniëerd moeten worden met betrekking tot technische haalbaarheid en algemeenheid. Aspekten, die hierbij een rol spelen, zijn of de benodigde technologie in zelfbouw ontwikkeld moet worden of dat deze op de open markt aangeschaft kan worden.
- *[vormgeving]* de wijze, waarop elk produkt op de open markt aangeboden zal worden, verdient de nodige aandacht. De inschatting van de konkurrentie op de markt speelt hierbij een wezenlijke rol, terwijl verder ook van belang is of op reeds bestaande ontwikkelingen ingespeeld kan worden.
- *[plaatsing]* de vormgeving van het produkt dient naadloos aan te sluiten aan de mogelijkheden op de markt. Belangrijk is daarbij de afschatting of een bestaande infrastruktuur zoals marketing organisatie of distributie–netwerk gebruikt kan worden.
- *[vraag]* voor elk produkt, waarnaar op de open markt nog geën vraag bestaat, dient deze vraag alsnog gecreëerd te worden. De kosten van marktontsluiting spelen voor de introduktie van het produkt een niet te verwaarlozen rol, mede in kombinatie met de wijze van marktbenadering.

In de getoonde differentiëring komt de paradoxale samenhang tussen technologie en marketing tot uitdrukking. Dit uit zich in de gebruikelijke innovatie–cyklus, waarbij vaak een produkt eerst als technologisch hoogstandje met moeizaam gebruik en grote verkoopsmarge geïntroduceerd wordt vooraleer het zich een weg baant naar de massa–markt met vergroot gebruiksgemak en slinkende marge. Vanuit de plaatsbepaling in deze view volgt dan ook direkt een voorspelling van het verdere leven, zowel in duur als in haalbare omzet c.q. winst. M.a.w. het maakt een afschatting mogelijk of en in hoeverre het zinvol is met MAGIC deze markt te betreden.

### 2.1.2 Interaktie

Bij de **interaktie–focus** is een aantal deelgebieden te onderscheiden die in hun nadruk lopen van de mens tot het gebruiksobjekt. Deze verbinding tussen enerzijds mensgerichte en anderzijds techniekgerichte studies geven het multi–disciplinaire (gamma) karakter van de bedrijfsmissie aan. Als voorbeeld van een gebiedsverdeling is te noemen:

- *[vorm]* het "Wohlempfinden" van de mens staat hierbij centraal: hoe wordt een vorm ervaren, en welke vorm weerspiegelt het beste de doeleinden van het te ontwikkelen gebruiksobjekt. Aangezien het voornamelijk om gevoelsmatige (psychologische) thema's gaat, is het niveau van interaktie van de mens met de virtuele omgeving nihil.
- *[funktionering]* het leren omgaan (trainen) van de mens met het objekt is belangrijk voor de vraagstellingen die gewoonlijk onder de noemer "Mens–Machine Interface" bestudeerd worden. Daarbij is naast de bovengeschetste statische konditie tevens de dynamiek van het omgaan met de virtuele omgeving van belang.
- *[funktie]* het optimaal omgaan van de mens met het objekt leidt tot vraagstellingen die b.v. onder de Arbeidspsychologie verder uitgewerkt worden. Hierbij is zowel de mens in zijn handelen als de omgeving in zijn virtuele bestaan volledig dynamisch en mogelijkerwijs zelfs onvoorspelbaar.

In de getoonde differentiëring komt een gang vanuit de Sociologie tot de Psychologie tot uitdrukking. Interaktie wordt technologisch mogelijk gemaakt op steeds goedkopere platforms (b.v. een 166 MHz Pentium–PC), maar de wijze van inrichting is nog volop in ontwikkeling. Zo wordt voor de samenstelling van "homepages" op de Elektronische Snelweg regelmatig verwezen naar de visualiserende artiest, wat aangeeft dat hier nog volop van kunst en nog weinig van kunde sprake is. Dit is tekenend voor een technologie in een (pre–) nataal stadium; men vergelijke bij voorbeeld met de analoog ontwerper in de mikro–elektronica van de jaren tachtig, de lay–out artiest in de IC–technologie van de jaren zeventig, de hacker

in de UNIX–wereld van de jaren tachtig, enz.. Deze paradoxale situatie wordt gewoonlijk eerst verbroken met de aanwezigheid van funktionerende systemen, waardoor laag–niveau optimaliteit geen ekonomische waarde meer heeft.

### 2.1.3 Objekt

Bij de **objekt–differentiatie** speelt het virtuele objekt een bepalende rol. Hierbij dient overigens een onderscheid gemaakt te worden tussen een specifiek objekt en het objekt geplaatst in een omgeving waarvan ook de mens deel uitmaakt. Als voorbeeld zijn de volgende thema's te noemen:

- *[architektuur]* Het gaat hierbij om de ruimtelijke ordening van de virtuele omgeving en het objekt als beleefd door de mens. Men denke b.v. aan de invloed van een landschap op het gebruik van de weg door de proefpersoon.
- *[implementatie]* Het gaat hierbij om de opbouw en aankleding van het virtuele landschap, waarin de proefpersoon geplaatst is. Men denke b.v. aan het ontwerp van een kantoorgebouw.
- *[realisatie]* Het gaat hierbij om de opbouw van een specifiek objekt binnen zijn omgeving. Men denke aan de samenstelling van een keukeninrichting door de koper voor zijn (hier gesimuleerde) woning.

In de getoonde differentiëring wordt de afstand tussen persoon en objekt stelselmatig verkleind. Dit heeft tevens gevolgen voor de eisen die aan de technologie gesteld worden. Immers, bij de architektuur zal de persoon zich in een groter geheel bewegen, zodat details in de uitvoering van het objekt niet van groot belang zijn. Naarmate de persoon zich minder kan abstraheren van de omgeving, nemen de eisen aan de visualisatie toe. In die volgorde is ook vaak sprake van een contemporaine technologische haalbaarheid.

### 2.1.4 Omgaan met het model

In het voorafgaande komt het beeld naar voren van een 3–dimensionaal arbeidsveld. Daarbij moet echter aangetekend worden, dat met name de objekt–differentiatie tot een veelheid aan inzetgebieden leidt. Binnen het raamwerk van de verkeerstechniek zien we de architektuur als de invloed van de omgeving (landschapsarchitektuur), de implementatie als de inrichting van een weg (civiele techniek) en de realisatie als de uitvoering van een kruispunt (electrotechniek); binnen het raamwerk van de kantoortechniek zien we architektuur als de invloed van de organisatie (bedrijfskunde), de implementatie als de inrichting van de informatiestromen (informatica) en de realisatie als de gebouwsamenstelling (bouwkunde).

Er bestaan dus vele 3–dimensionale kubussen, waarop MAGIC zich kan richten. Om hierin tot keuzes te komen is minstens een verdere kwalificatie (en liefst zelfs een kwantifikatie) noodzakelijk. Voor elk 3–dimensionaal punt kan in eerste instantie met een cijfer tussen 0 en 10 (0=slecht; 10=uitmuntend) de verwachting uitgesproken worden, in hoeverre de korresponderende vectorvelden reeds dicht bij een ekonomische exploïtatie liggen. De gegeven numerieke beoordeling is daarbij slechts van marginale interesse. Het gaat veeleer om de resulterende weergave van trends en daarmee van mogelijkheden om vanuit de te creëeren technologische basis op meerdere terreinen actief te worden.

Desondanks zijn bij verdere uitwerking kwantitatieve gegevens wel bepalend voor de taktische uitwerking van de bovengeschetste strategische waarneming. Van meer dan bijzondere belang zijn de kosten om een bepaald niveau te bereiken, gekoppeld aan de potentiële baten die op hun beurt zijn af te leiden aan de marktgrootte en de daaraan gekoppelde haalbare winstmarge. Daarbij dient vanwege het hoog technologische karakter van de produkten aandacht gegeven te worden aan de bijzondere positie van de "launching customer". Ofschoon daarmee reeds een specifieke markt bediend wordt, zal het vanuit de veranderende bedrijfsmissie toch onontkoombaar blijken om extra te investeren, waarbij vanuit een goede markt–analyse de verwachting geschapen dient te worden dat deze gelden binnen 2 jaar onder toenemende konkurrentie terug verdiend kunnen worden. M.a.w. de hier gebezigde analyse zal goed ondernemerschap niet kunnen vervangen.

In het volgende zal een poging gedaan worden om enkele marktsegmenten die in eerste zicht tot het werkgebied behoren te verkennen. Daarbij beperken we ons tot een viertal mogelijkheden: (a) de vervoerssektor, (b) de winkelsektor, (c) de kantoorsektor, en (d) de woninginrichtingssektor, waarbij vrij willekeurige voorbeelden van potentiële produkten genomen worden. Deze worden als karakteristiek gezien voor de mogelijkheden, zoals die zich op korte en middel–lange termijn kunnen voordoen.

## 2.2 Funtioneren in een omgeving

In deze sektie zal voornamelijk aandacht geschonken worden aan situaties waarin het gedrag van de mens beïnvloed wordt. De aspekten daarvan reiken van veiligheid in handelen tot regelrecht sociaal gedrag. Het eerstgenoemde aspekt komt hierbij slechts zijdelings aan de orde, maar dit is geen indikatie van het konkrete belang. Immers, veiligheid geeft weliswaar geen direkte verkoop, maar levert wel de mogelijkheid om betreurenswaardige omstandigheden en daarmee samenhangende kosten te besparen.

In vele industriële situaties is de reaktie van de mens onder onvoorziene omstandigheden afhankelijk van de mate waarin deze omstandigheden voor hem duidelijk gemaakt worden middels zulke abstrakte aanwijzingen als rode, flikkerende lampen en wijzeruitslagen. Plaatsing en vormgeving zijn dan bepalend voor de mogelijkheid dat later van een "menselijke fout" gesproken moet worden. Ofschoon met de "mens" in deze kenschetsing dan de operator bedoeld wordt, lijkt het redelijker dat de ontwerper van het indikatie–platform de schuld moet dragen.

### 2.2.1 Vervoersektor

In het bijzonder richten we ons hier op de inrichting van de civiele infrastruktuur. Als produktidentificatie poneren we hier: de *advisering op de samenhang tussen de landschapsarchitektuur, de wegstruktuur en de verkeersveiligheid*. We gaan in eerste instantie uit van de volgende invulling. Bij architektuur denken we aan de mens op een bekende weg in een bekend landschap. De vraag is dan hoe de inrichting van het landschap moet zijn om een goed rijgedrag te bevorderen, d.w.z. om de attentie van de bestuurder niet af te leiden. Bij implementatie wordt zulks uitgebreid met verschillende trajecten van uiteenlopende moeilijkheidsgraad, teneinde de bestuurder gericht te trainen op het omgaan met moeilijke situaties. Tenslotte kan gedacht worden aan de bestuurder in zijn omgang met een nieuw te ontwerpen voertuig. Daarbij is van belang om het ontwerp van het virtuele voertuig te optimaliseren met betrekking tot zaken als veiligheid, gezondheid, stress etc.. In ieder van deze kategorieën kan een verschillende hoeveelheid interaktie verondersteld worden.

De vorm van een landschap is reeds nu in gebruik; m.b.t. de weg alsmede de kruising is enig aanvullend raffinement mogelijk gewenst. Aanvulling van de gebezigde vormen voor passieve funkties vergt relatief weinig extra werk, terwijl voor een werkelijk interaktief uitoefenen van de funkties nog een wezenlijke bijdrage noodzakelijk is. Aan de vraagkant ziet de situatie er gemengd uit. Voor een eenvoudige vorm zal weinig rek bovenop de huidige vraag bestaan. Voor het passief reageren op verkeer is extra interesse van verzekeraars mogelijk; voor de actieve uitvoering kan zowel bij privé–huishoudens als bij (toeleveranciers van de) automobielindustrie belangstelling bestaan. Op grond van deze overwegingen komen we nu (zij het arbitrair) tot de volgende kwalitatieve invulling.

Met enige voorzichtigheid kan hieruit gekonkludeerd worden, dat op de bestaande markt reeds enige zaken gedaan kunnen worden, hoewel aan de technologische kant niet alles marktgereed is. Een verdere uitbouw zal met name technologie gericht dienen te zijn, terwijl op langere termijn aan een degelijke marktbewerking niet te ontkomen valt. Daarbij valt op, dat in de richting van de verdere opvoering van de interaktiviteit zelfs gesproken mag worden van een duidelijke onderzoeksbehoefte; de produkt–definitie ligt daar nog zover weg, dat van een directe utilisatie nog niet gesproken mag worden.

In deze sektor hebben op de achtergrond steeds een tweetal soorten klanten gespeeld: de Overheid en de Automobielindustrie. Daarbij zal de Overheid eerder in de architektuur en eventueel implementatie geïnteresseerd zijn, terwijl de aandacht van de Automobielindustrie verwacht mag worden bij de realisatie en eventueel implementatie. Overwegend dat de Overheid niet direkt een opdrachtgever van allure is, ter-

wijl de Automobielindustrie een sterk op zichzelf gericht karakter heeft, gaat onze voorkeur niet direkt uit naar dit applikatie–gebied.

### 2.2.2 Winkelsektor

In het bijzonder richten we ons hier op grootwinkelbedrijf, waarbij de exploitatie van de winkel als verkooppunt geoptimaliseerd dient te worden. Als mogelijk produkt poneren we hier: *de advisering op de samenhang tussen winkelaankleding, inrichting en routering*. De definitie van een produkt heeft een samenhang met bestaande simulatie–spelen, met dien verstande dat de afbeelding niet uit verkeerselementen maar uit schappen en winkelprodukten bestaat. Wat blijft is de navigatie door een virtuele omgeving met een mate van interaktie met mede–gebruikers. Deze marktsektor kent een groot gebruik van informatie–technologie (met name in logistiek en planning), zodat de ontvankelijkheid groot is; verder zijn sonderingen naar virtuele produkten merkbaar, maar dit heeft nog niet tot concrete vragen aanleiding gegeven.

Als we nu pogen tot een nadere invulling te komen, kan bij architektuur gedacht worden aan de algemene aankleding van de winkelruimte. Een winkel is een openbare ruimte, die gericht een geselecteerd publiek dient aan te spreken en uit te nodigen. Een zekere sociale funktie is onmiskenbaar aanwezig. Bij de implementatie speelt de specifieke inrichting in produktsoorten en schapbezetting een rol. Klantvriendelijkheid gaat nu over in een winkelvriendelijkheid van de klant: een zeker winkelpatroon moet vanzelfsprekend zijn. Tenslotte voegt de realisatie de verblijfsaspekten van de klant toe. Dit is met name van belang voor de real–time bepaling van het aantal kassa's.

Wat betreft de vorm vergt de routering niet alleen een uitbreiding van de kollektie landschapselementen, maar verder vindt er ook een duidelijke vermindering van het blikveld plaats, zodat de eisen aan de afbeeldingssnelheid bij het "ronden van hoeken" zullen toenemen. Dit is vergelijkbaar met de verkeerssektor op het moment dat stadstaferelen aan de orde komen. De waardering van het passieve funktioneren zal opnieuw opgezet moeten worden, maar de behandeling van de interaktie heeft veel gelijkenis met het voorgaande geval. Aan de vraagkant moet opgemerkt worden, dat hier waarschijnlijk alleen aan het grootwinkelbedrijf gedacht moet worden. Gezien de volwassenheid van deze markt kan eigenlijk alleen op extra belangstelling voor de interaktieve versie gerekend worden. Op grond van deze overwegingen komen we nu (zij het arbitrair) tot de volgende kwalitatieve invulling.

Voor de winkelsektor speelt eigenlijk alleen het grootwinkelbedrijf of de belangenorganisaties een rol als afnemer. Hoewel de technologische problemen niet werkelijk groot zijn, is er wel veel werk te verrichten voordat er een versie ontstaat die meer brengt dan wat reeds voor handen is. Verder lijkt de markt maximaal zo klein als bij de vervoerssektor. Ook op langere termijn zijn de vooruitzichten niet imponerend: er bestaat geen konkrete vraag, er bestaat geen direct technologische aansluiting en de omzetverwachting is twijfelachtig. Oftewel, een marktsektor, die eerder als een off–spin dan als een target gezien moet worden: leuk om er zo even bij te nemen, maar zinloos om er speciale moeite voor te doen.

## 2.3 Leven in een omgeving

Leven in een omgeving legt een nadruk op de meubilering. In zijn algemeenheid kunnen daarin drie sektoren onderscheiden worden: (a) de woonmeubelindustrie, die een kleine 2% groei in omzet kent met een afzet die zich voor meer dan 80% via speciaalzaken afspeelt, (b) de bedrijfsmeubelindustrie die hoewel nog steeds van respektabele omvang (730 Mfl in 1993) zich een 5 tot 10% daling moet getroosten, en (c) de interieursektor die relatief konstant blijft (d.w.z. een 0.5% daling). In het volgende nemen we ons enige voorbeelden uit dit brede palet.

### 2.3.1 Kantoorsector

In het bijzonder richten we ons hier op de projektontwikkelingsproblematiek zoals die speelt bij makelaars op de zakelijke markt. Als mogelijk produkt poneren we hier: *de advisering op de samenhang tussen business design, informatie–technologie en het kantoorbeheer.* De definitie van een produkt heeft ook hier

weer een samenhang met de spel–simulator, met dien verstande dat de afbeelding uit delen van een kantoor interieur bestaat. Wat blijft is de navigatie door een virtuele omgeving met een mate van interaktie met mede–gebruikers. Deze marktsektor kent een relatief gering gebruik van informatie–technologie (hier is drukmateriaal nog erg populair), zodat de ontvankelijkheid niet groot is; een vraag zou gestimuleerd kunnen worden bij (met name) telematica gebouwen, i.e. gebouwen die met informatie–technische infrastruktuur aangeboden zullen worden.

Als we nu pogen tot een nadere invulling te komen, kan bij architektuur gedacht worden aan de algemene aankleding (huisstijl) van het gebouw. Deze moet non–verbaal de missie van de onderneming zichtbaar maken. Bij implementatie wordt nu gedacht aan de kantoor–indeling als weergave van de organisatie struktuur van de onderneming, waarbij efficiency en werkklimaat vooropstaat. Verder laat realisatie zich zien als de informatie–technische infrastruktuur van het kantoor als verdere invulling en ondersteuning van de organisatie struktuur. In ieder van deze kategorieën kan een verschillende hoeveelheid interaktie verondersteld worden.

De problematiek in deze sektor lijkt in eerste opzet tussen de beide voorgaande in te liggen. Nieuwe landschapselementen zijn nodig, maar liggen dicht bij een stadslandschap. Voor de integratie van de beoogde sociale organisatie zal ook een zekere vernieuwing moeten plaatsvinden, terwijl voor de werkfunktie nog een portie onderzoek noodzakelijk is. Een verdere uitbreiding naar de waardering voor passief funktioneren lijkt dicht bij de vereisten bij de verkeerssektor te liggen, terwijl actief funktioneren een nog intensere interaktiviteit zal laten zien. Aan de vraagkant is momenteel weliswaar nog geen behoefte te bemerken, maar het geringe technologische gat laat bevroeden dat een aansluiting op termijn goed haalbaar moet zijn. Op grond van deze overwegingen komen we nu (zij het arbitrair) tot de volgende kwalitatieve invulling.

Vanuit de reeds aangegeven plaatsing tussen de beide voorgaande voorbeelden lijken hier gerede mogelijkheden voor een gezonde ekonomische exploitatie te liggen. De potentiële klanten zijn kapitaalkrachtig en technologie–gretig, zodat een aansluiting met geringe middelen mogelijk moet zijn. Een verdere uitbouw zal met name technologie gericht worden, terwijl op langere termijn aan een degelijke marktbewerking niet direkt noodzakelijk hoeft te zijn. Daar staat echter tegenover, dat het in deze sektor om totaal anders gerichte klanten gaat. Technologie–gretigheid gaat hier vaak hand–in–hand met technologisch analfabetisme, zodat hier om een grote ondersteuningsbehoefte gerekend moet worden.

In de kantoorsektor zijn in tegenstelling tot de woningsektor niet zoveel spelers. Ofschoon in eerste instantie de kleine makelaar niet uitgesloten hoeft te worden, is niet direct in te zien hoe hier een vraag gestimuleerd kan worden. De huidige situatie op de kantoormarkt kenschetst zich door een struktureel overschot. Momenteel is omstreeks 34 $Mm^2$ kantooroppervlak beschikbaar, waarvan een 10% voor langere tijd leeg staat. Het is duidelijk dat de lokatie een grote rol speelt bij de uiteindelijke keuze. We zien de rol van MAGIC dan ook eerder bij de verkoop van kantoren, waarvan de lokatie niet direkt bevorderlijk is voor de verkoop.

### 2.3.2 Woninginrichtingssector

In het bijzonder richten we ons hier op de directe verkoop van kapitaalgoederen op de burgermarkt. Als mogelijk produkt poneren we hier: *de advisering op de samenhang tussen binnenhuisarchitektuur, leefklimaat en gebruiksgemak*. Deze sektor lijkt in eerste instantie nog het verst af te staan van de gebruikelijke spel–simulaties. Overigens blijven de wezenlijke kenmerken nog steeds bewaard: ook hier zal sprake zijn van navigatie door een virtuele omgeving, maar zal de wijze van interaktie anders van karakter kunnen zijn. Ook deze marktsektor kent een relatief gering gebruik van informatie–technologie; stimulering van de vraag zal sterk afhankelijk zijn van infrastrukturele kondities zoals de uitgebreidheid van het kabelnet. Ofschoon de wezenlijke gebruiksvoordelen bij individuele klanten liggen, zal ook hier voor "launching customer" eerder ook een grote leverancier zoals Wehkamp gedacht moeten worden.

Als we nu pogen tot een nadere invulling te komen, kan bij architektuur gedacht worden aan de inrichting van ruimten in woningen, zoals de woonkamer. In Nederland is dit meestal de grootste ruimte in een woning, en tevens de ruimte met enige grote, duurdere meubelartikelen. De keuze van een specifiek meubel–

element is gewoonlijk niet rationeel, zodat ruimte voor het "beleven" een nuttige funktie heeft. Bij implementatie zal deze keuze behoefte verder aanleiding geven tot het verschuiven van meubelen, wandaankleding e.d.. Bij realisatie zullen de zelfbouwmeubelen samengesteld worden. Overigens is in dit kader ook aan de samenstelling van een keukeninrichting te denken. In ieder van deze kategorieën kan een verschillende hoeveelheid interaktie verondersteld worden.

Als illustratie van de mogelijkheden in deze marktsektor, die nogal klassiek van klantbenadering en daarom weinig innovatie–behoeftig aandoet, noemen we het FLY–koncept. Dit Franse concern is pas enkele jaren geleden van start gegaan. Vanuit een sterk automatiseringskoncept is direkt ingesprongen op de markt voor "ready–wear" meubels. De onderliggende informatie–technologie stelt FLY in staat om de bewegende behoeftes in de markt te meten en direkt voor de bestaande markt te produceren. Dit koncept doet sterk aan de MAGIC filosofie denken en het lijkt daarom redelijk aan te nemen, dat in de woninginrichtingssektor goede resultaten te behalen zijn.

De problematiek in deze sektor ligt van alle genoemde alternatieven nog het dichtst bij de verkeerssektor, zij het dat bij de woninginrichtingssektor de gebruiker ook inbreng in de samenstelling van het "landschap" heeft. Er treden hier dus extra vormen van interaktiviteit op. Dit leidt ertoe dat voor de bewerking van de markt waarschijnlijk uitgegaan moet worden van de laag–voor–laag bewerking met grote aandacht aan voor de marktbediening noodzakelijke infrastrukturen. Verder zullen als potentiële afnemer alle marktpartijen in aanmerking komen, maar daarbij wel ieder eigen wensen (en dus eigen produkten) hebben. Op grond van deze overwegingen komen we nu (zij het arbitrair) tot de volgende kwalitatieve invulling.

Met enige voorzichtigheid kan hieruit gekonkludeerd worden, dat met het bestaande produkt reeds enige zaken gedaan kunnen worden, hoewel aan de technologische kant niet alles marktgereed is. Een verdere uitbouw zal met name technologie gericht dienen te zijn, terwijl op langere termijn aan een degelijke marktbewerking niet te ontkomen valt. Daarbij valt op, dat in de richting van de verdere opvoering van de interaktiviteit zelfs gesproken mag worden van een duidelijke onderzoeksbehoefte; de produkt–definitie ligt daar nog zover weg, dat van een directe utilisatie nog niet gesproken mag worden.

# Chapter 3 ITS MAGIC specified

This chapter describes the formal specification of the system described in the first chapters. Instead of starting immediately with this formal specification a summary of the system to be specified is given. The subsequent section will be a brief description of the way we developed, designed and implemented a part of ITS MAGIC. In the following section the specification method used will be described. After this description the results of the formal specification of ITS MAGIC are presented. The chapter will end by drawing some conclusions from the formal specification.

## 3.1 ITS MAGIC in summary

The explanation of the acronym ITS MAGIC is: Interactive Tele–Shopping with MediA Guided Intelligent Computing. This means that we like to develop a system with which clients can shop at a supplier while sitting at home on their own couch. Instead of visiting annoying, awfully busy shopping centers or just reading advertising material at home, the client can shop interactively from his couch at any time he wants to. The interaction between the client and the system means, that the actions of the client will cause reactions of the system, and vice versa. The actions and reactions of the system will be made "visible" by using several media (multi–media), like sound and video. We also like the system to treat every person as a special, unique human being, with his own interests, behavior etc.. This is the intelligent part of ITS MAGIC. When the previous items will be realized, the clients will feel confident and safe while working with the system. Due to this positive feeling the client likes to work with the system and he[1] will probably order more products than working with a system which treats all the clients in the same way.

As can be read in chapter 1, the client will be connected to the supplier over a network. By using this direct connection the supplier can inform the client with the most actual information at any time. Instead of sending all potential clients a new floppy or CD–ROM with the newest products, the new commercial slogans, new logos etc., the supplier only has to update his own databases and other concerning configurations. The supplier will also be able to receive and process orders or client information directly. In this way the supplier can react directly at changes occurring in the marketplace.

So the basis for the system is settled, we've got clients, providers and a network. To develop a system that is, in our view, *useful, successful and marketable*, we placed some demands on the system. The key demands on ITS MAGIC are:

- speed
- adaptivity

1. In this report 'he' should be read as 'he' or 'she'.

- platform and application independency
- re–usability

These demands will be described briefly in this section. In the following chapters the demands are elaborated in more detail. Therefore the following descriptions will refer to the sections, in which the demands will be implemented.

**speed**

When a client contacts a producer he wants to have a quick response, because he hates to wait longer than a few seconds. So when discussing ITS MAGIC it's necessary that the sales talk of the producer starts as quickly as possible. This can be done by sending only the settings of the sales talk to the client, instead of sending a large, bandwidth consuming, and in many cases also time consuming, "movie". The user program, installed at the home of the client, has to build and interpret the sales talk using the received settings. Because this last process can be a heavy one, the client has to have a computer with decent processing power. More detailed information will be given in section 4.2.

**adaptivity**

If a producer wants to sell a product to a client, he would be wise to treat a client in a personal manner. In this way the producer keeps the client satisfied and the producer will be able to sell as many products as possible. We want to use this property in ITS MAGIC. In other words, the user program has to be adaptable to a certain client. When doing so the client will get personally involved and feel safe and comfortable. In this way he will probably buy products easier than when feeling bad and upset.

The user program can't be a hard–coded program for a certain producer. It has to be adaptable to every producer in the world. So the client needs only to have one program to be able to shop at every producer he wants to.

**platform and application independency**

If a producer wants to reach a lot of potential clients, the user program has to be able to run on different systems. This means that the program has to be able to run on different platforms (platform independence) and that it should be possible to replace any application (e.g. a 3D rendering engine) with a newer version of the same application or even with another application (application independent). This can be done by introducing a layered structure. More detailed information will be given in section 4.3.3.

**re–usability**

To lower costs it is necessary to reuse software and data as often as possible. On the one hand a lot of the software developed for the user program has to be reused for developing the producer/middleman system (see section 3.5). On the other hand, the supplier should not have to develop a completely new system, but (re–)use generic components, which he can adjust by setting some configurations (see also chapter 6).

## 3.2 Nothing to something

When we started this project we had a very rough idea of what to do. We wanted to make a useful interactive, intelligent selling application. After some brainstorm sessions and searches in libraries for ideas, we came to the ideas and demands given in the previous summary. At the moment that the ideas were settled the system had to be specified. After the production of some informal specification documents, the specification had to become formal and documented clearly. This chapter contains the results of the formal specification using SADT.

As an alternative, we also looked at the specification language RM–ODP. We decided not to use RM–ODP, because the language was mainly created for (large) distributed systems with lots of performance

and safety constraints. Although the ITS MAGIC system is a distributed system, it is not really distributed in terms of RM–ODP, nor does it have real–time constraints. In short, using RM–ODP would be overkill in our situation. For more information about RM–ODP, we refer the reader to [5], where a complete definition of the specification language can be found.

After completing the specification, we had to decide which part of the system we wanted to implement. We have chosen for the implementation of the user part of ITS MAGIC. For the underlying motivations of this decision see the conclusions at the end of this chapter (section 3.5). Using the formal specification and the demands placed on the system (see chapter 1 or section 3.1), the ITS MAGIC system architecture was developed (see chapter 4). By determining the formal specification the required (main) elements of the systems were defined. By using the specification, it was also possible to define the interfaces between the elements.

From that moment on, the real implementation got started. The first phase was the translation from specification and definition to implementation (e.g.. data–types were translated into data–structures, the elements were structured etc.). During the second phase the programming code was produced and the formal specification and the definitions were slightly modified as the result of new insights.

## 3.3 Introduction to SADT

The main topic of this chapter is the formal specification of the system specified previously in informal language. The specification method used is SADT. SA (Structured Analysis) is a graphical specification language developed in the seventies by Douglas T. Ross [1]. SADT (Structured Analysis and Design Technique) is based on this method and developed by Ross and his colleagues of SofTech. A brief introduction of the language and the way of thinking will be described next.

The SADT graphic language provides a limited set of primitive constructs from which analysts and designers can compose orderly structures of any required size. The notation is quite simple. **Boxes** represent parts of a whole in a precise manner. **Arrows** represent interfaces between parts. **Diagrams** represent wholes and are composed of boxes, arrows, natural language names, and certain other notations. An SADT model is an organized sequence of diagrams, each with concise supporting text. A high–level overview diagram represents the whole subject. Each lower–level diagram shows a limited amount of detail about a well–constrained topic. Further, each lower–level diagram connects exactly into higher level portions of the model, thus preserving the logical relationship of each component to the total system. So an SADT model is a graphic representation of the hierarchic structure of a system.

SADT is derived from the way our minds work, and from the way we understand real–world situations and problems. Systems consist of things that perform activities that interact with other things, each such interaction constituting a happening. A functional architecture is a rigorous layout of the activities performed by a system and the things with which those activities interact. SADT is based on a functional architecture. An SADT model has to consider both the things and happenings of the subject being modeled. Happenings are represented by an activity decomposition consisting of activity diagrams (actigrams) and things are represented by data decomposition consisting of data diagrams (datagrams) The neat thing about SADT is that both of these complementary but radically different aspects are diagrammed using exactly the same notation (see Figure 3.1). The happening/activity and thing/data domains are completely dual in SADT. The two views taken together give a means of checking accuracy and completeness.

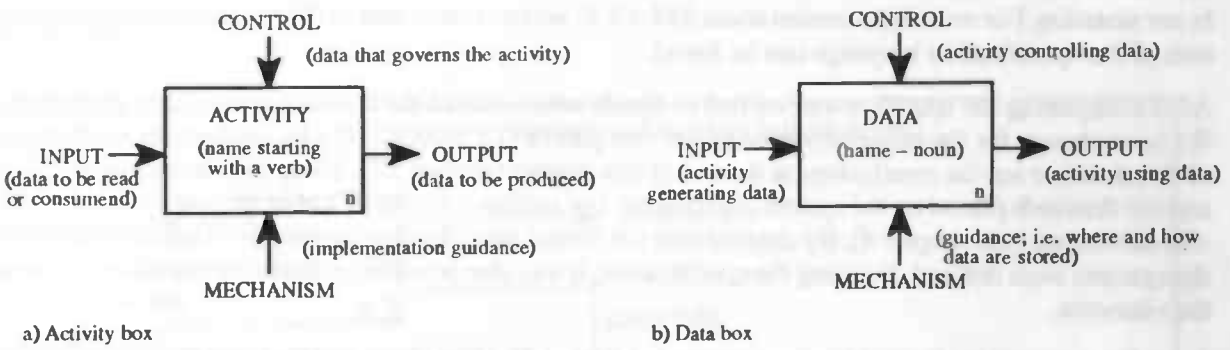



**Figure 3.1**: *Building blocks in SADT*

## 3.4 Using SADT

While trying to document the specification of the system properly, some notions have to be agreed upon. To clarify the decomposition of the system every box at each level has it's own number to identify itself. When a box (e.g. no. 2) at the highest level decomposes into, for example, four boxes, the numbers of the new boxes will be concatenated at the number of the "parent" box, separated by a dot (e.g. 2.1 ... 2.4). The number of a datagram will be preceded by a "d" and the number of an actigram by an "a". The datagram will also show the relation M:N between the data types. Figure 3.2 gives an example of how to use such a relation.


**Figure 3.2**: *A car has N different wheels, a wheel belongs to only one car*

To keep the diagram clear of redundant information, the arrows above (control) and below (mechanism) will only be shown when this is non–trivial. In some of the diagrams bold dots will appear while lines are intersecting. When a bold dot is used the lines really divert or convert. When no dots are used while lines are intersecting, these lines don't really intersect.

In this report each actigram will be succeeded by a short description of the contents of the actigram. The short description will be followed by the definitions of all the data and activity terms occurring in the actigram. The definitions will be succeeded by the datagram belonging to this actigram. Because the terms of the datagrams are (almost) similar to the terms of the actigrams the definitions are explained only once. The datagram will also be followed by a short description.

The purpose of the way the specification is documented, is that every level has to be self–explaining. The reader has to be able to understand a level without knowing the decomposition of other (higher) levels. In this way a level can be discussed without bothering about, at this level, non–important facts.

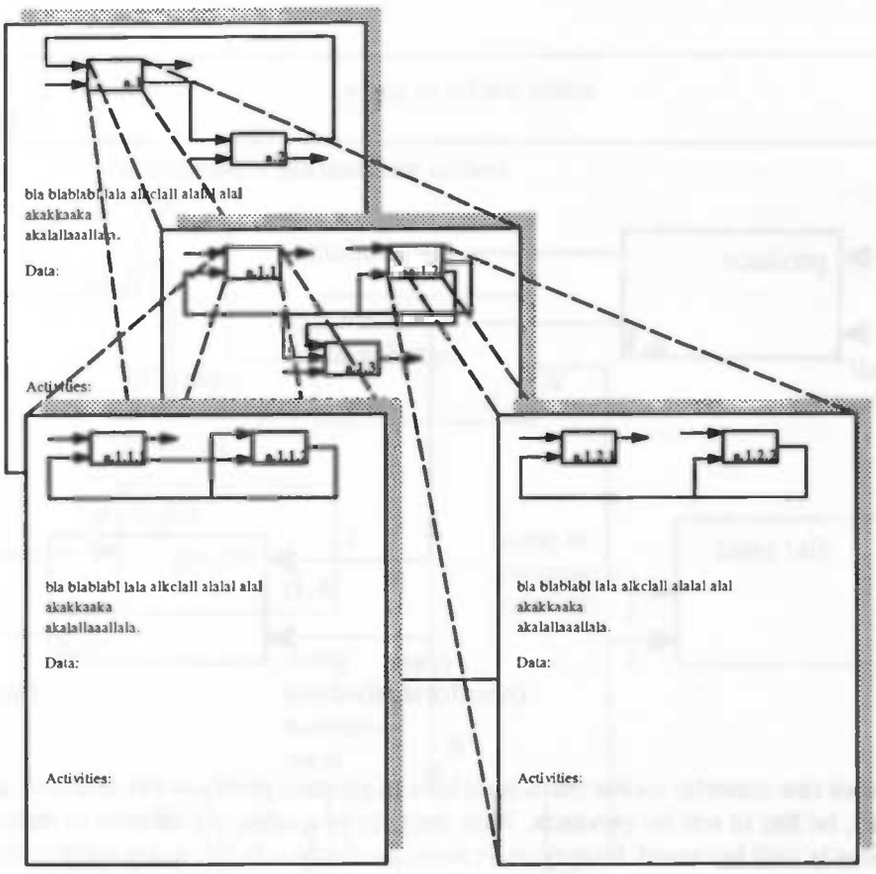
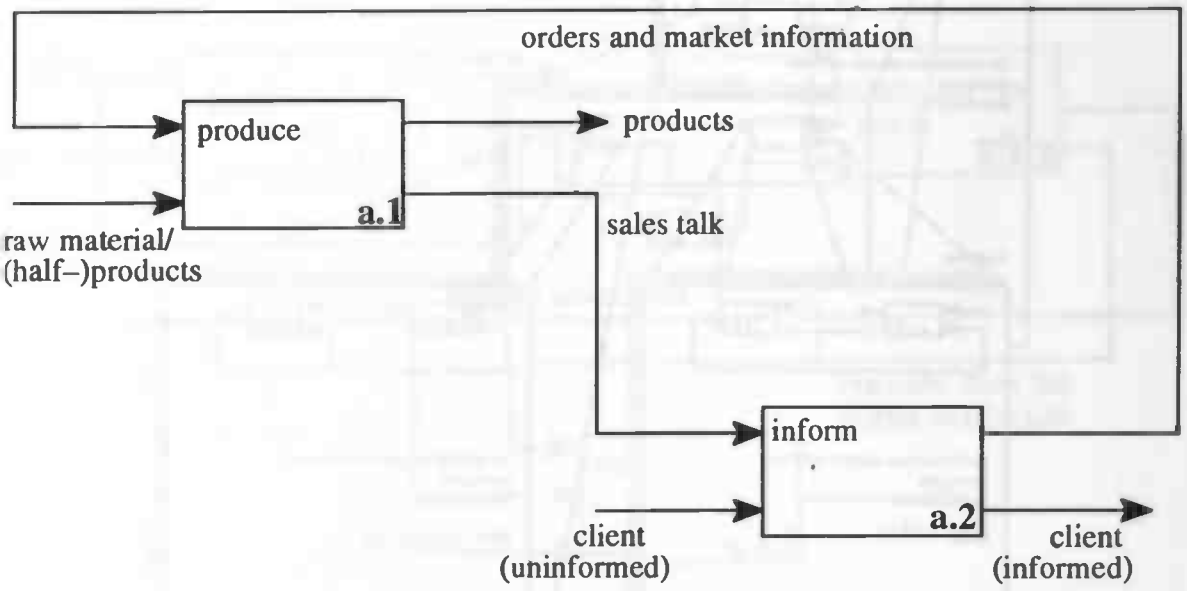



**Figure 3.3**: *The SADT hierarchy*

**level 0:**

**Actigram**




A producer uses raw material and/or (half–)products to produce products (1). Because a producer wants to earn money, he has to sell his products. With the help of a sales talk he tries to inform the client (2). When the client is well informed, he may order some products. It is necessary for the producer to receive enough information of the current market to be able to offer marketable products. So the producer likes to receive the reactions on his current products and other wishes, not necessarily concerning his own products, of his clients.

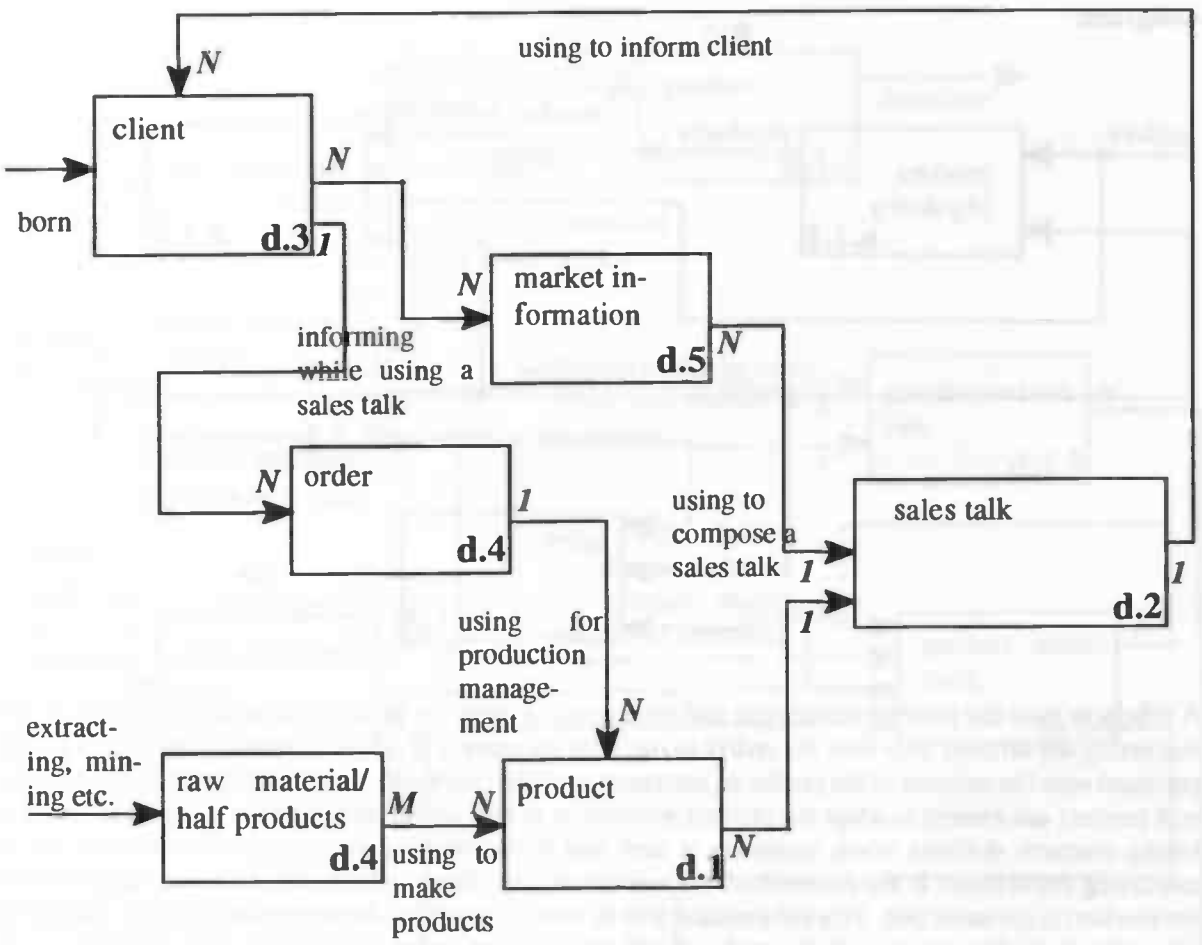
Data:

1. *(input)* **raw material/(half–)products:** needed for the physic production of the products.
2. *(input)* **client:** a subset of the market, to whom the producer wants to sell his products.
3. *(internal)* **sales talk:** the actions and reactions specified by the producer by which the client will be informed about the products of the producer.
4. *(internal)* **orders:** the products bought by a client.
5. *(internal)* **market information:** personal and behavioral information of the client.
6. *(output)* **client:** a subset of the market, who delivers the producer market information and maybe some orders.
7. *(output)* **products:** merchandise of the supplier, which he has to earn money with.

Activities:

1. **inform:** the client will be informed by a sales talk about the products of a producer and will deliver the producer market information and maybe some orders.
2. **produce:** the producer produces products out of raw material and/or (half–)products. He also produces a sales talk out of the orders and market information received from clients.

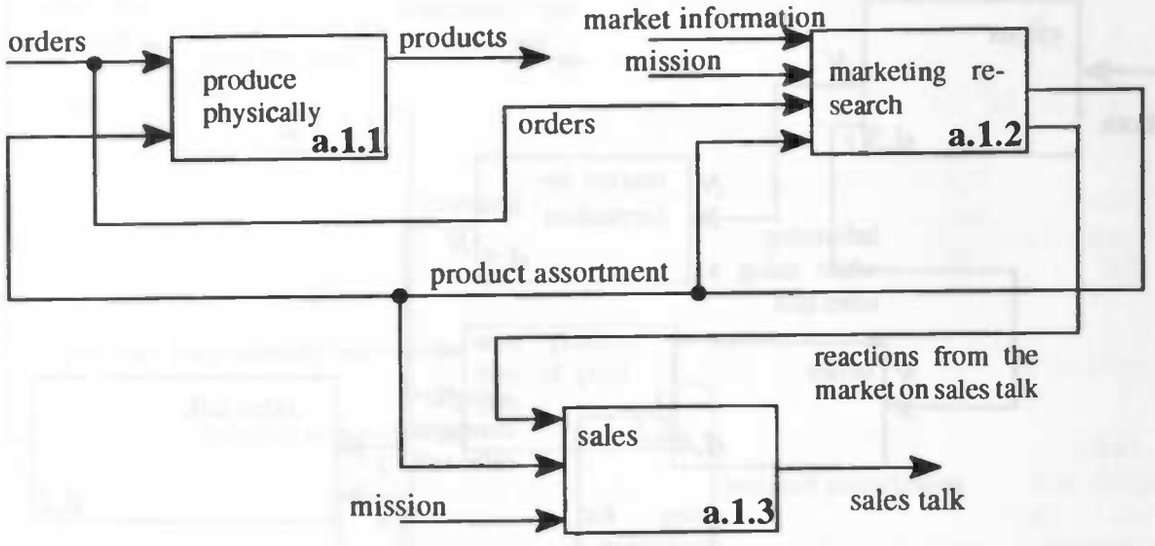
**Datagram:**




The data–type product arises from the processing of the raw materials and/or (half–) products. This will (only) be done when there are (enough) orders for a specific product. The sales talk will be made of the information about the products and the market information. The market information is generated by the client. The client also generates the orders. He only generates orders and market information as a reaction on a sales talk. The client data type arises from a natural process between two instances of the client data type.

**level 1: decomposition of *produce (a.1)***

**Actigram:**




A producer uses the product assortment and the incoming orders to produce the real products (**1.1**). The marketing department also uses the orders to research the market (**1.2**). In addition to the orders this department uses the mission of the producer, information of the clients (the market information) and the current product assortment to adapt the product assortment to the current wishes of the clients. So, the marketing research delivers when necessary a new and better product assortment. Another task of the marketing department is the research of the reaction of the clients, which results in the reactions from the market on the sales talk. This information will be used by the sales department to create or change the sales talk (**1.3**). The mission of the producer will also be used during the creation and adaptation of the sales talk.

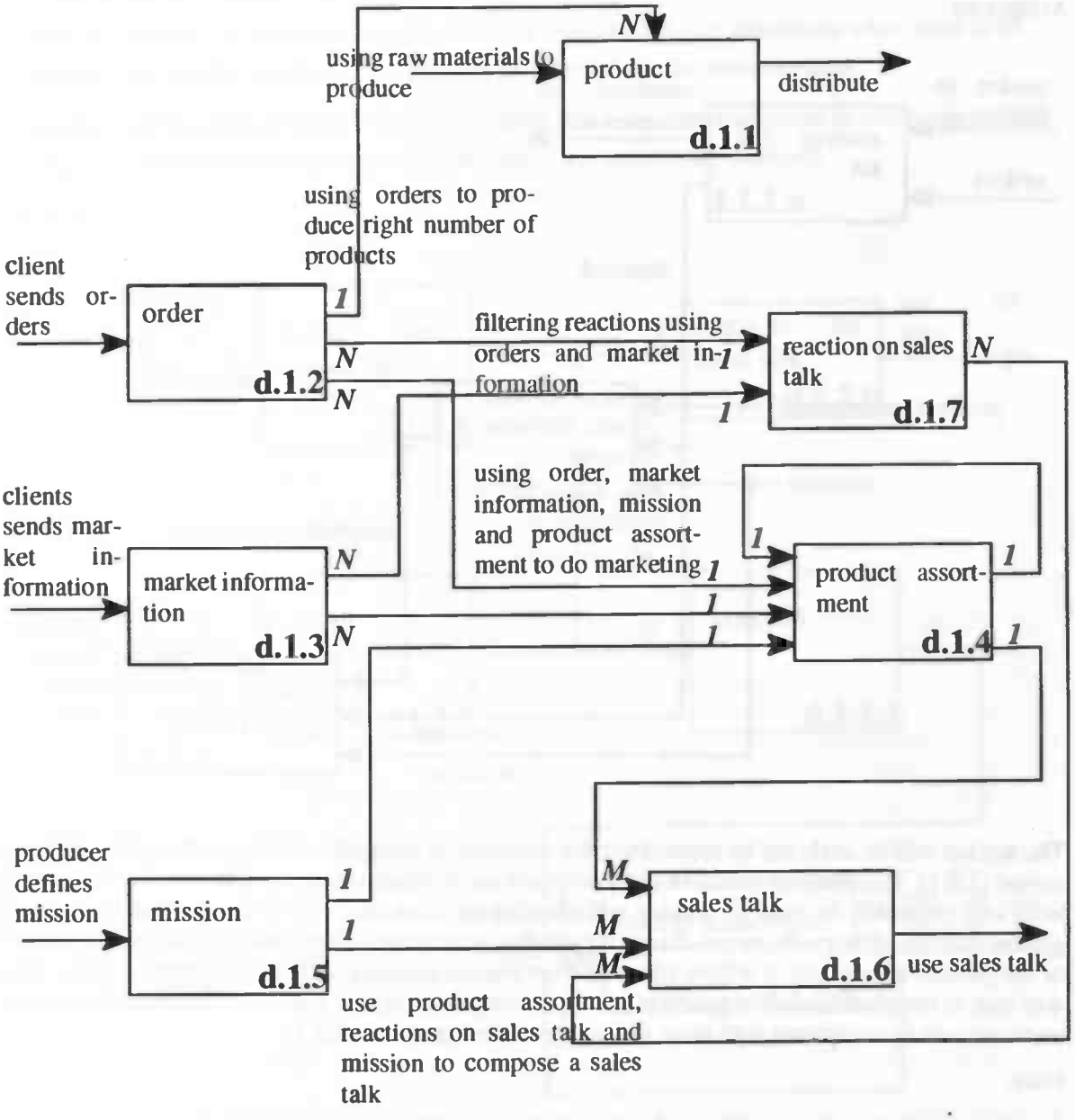
Data:

1. *(input)* **mission:** the way the producer wants to be and behave on the market.
2. *(input)* **orders:** the products bought by a client.
3. *(input)* **market information:** personal and behavioral information about the clients.
4. *(internal)* **product assortment:** all the products the producer produces.
5. *(internal)* **reactions from the market on sales talk:** information acquired by analyzing the reactions of clients on the sales talk and by analyzing the orders of specific clients.
6. *(output)* **sales talk:** the actions and reactions specified by the producer by which the client will be informed about the products.
7. *(output)* **products:** merchandise of the supplier, which he has to earn money with.

Activities:

1. **produce physically:** the real production of the ordered products.
2. **marketing research:** researching the market information and adapting the product assortment.
3. **sales:** creating and adapting the sales talks.

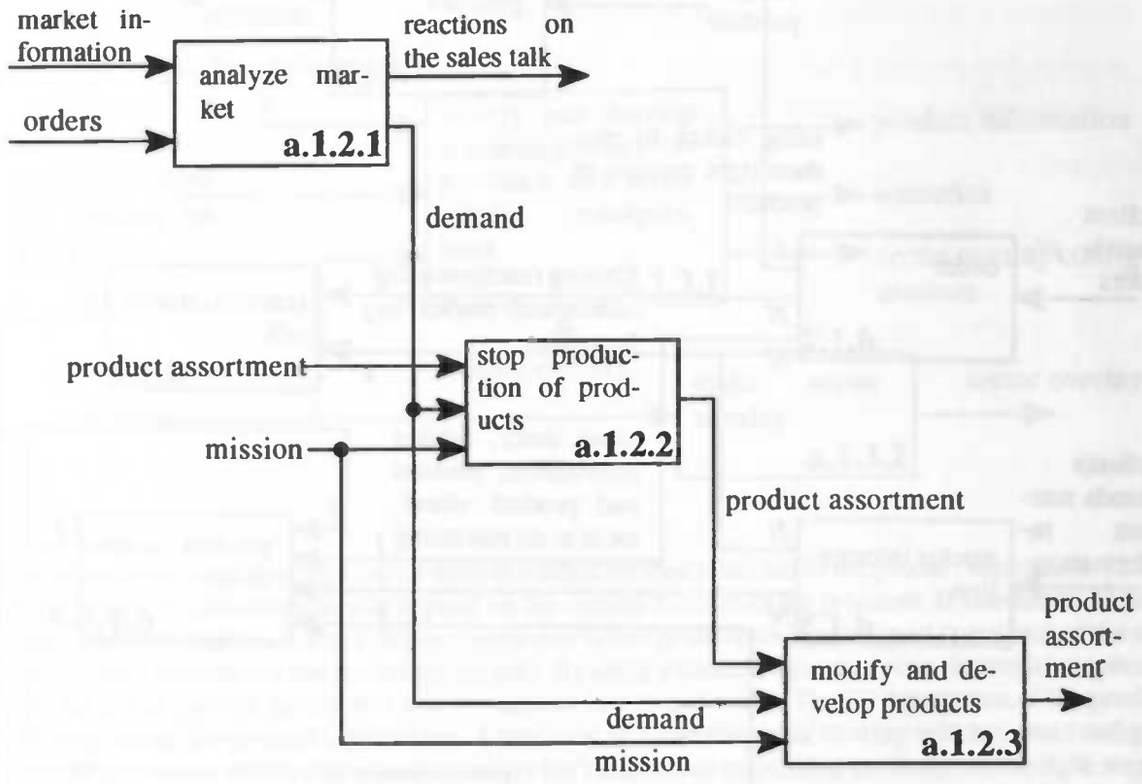
**Datagram:**




Instances of the data type product **(d.1.1)** will be made of raw materials and (half–)products. Instances will only be produced when several orders **(d.1.2)**, which are generated by the clients, exist. The instances of this data type will be distributed and eventually brought to the client. An instance of the data type reaction on the sales talk **(d.1.7)** will be generated by filtering the incoming orders and market information **(d.1.3)** from the clients. A product assortment **(d.1.4)** will be composed and/or adapted by using the orders, market information, the mission **(d.1.5)** of the producer and of course by using the current product assortment. This product assortment will be used, together with the mission, to make a sales talk **(d.1.6)**. This sales talk will eventually be used to inform the clients.

**level 2: decomposition of *marketing research (a.1.2)***

**Actigram:**




The market will be analyzed by researching the information about the clients and the orders that are received **(1.2.1)**. This analysis results in client demands and in reactions on the sales talk. These client reactions will eventually be used for making and adapting the sales talk. The demand of the market and the current mission of the producer can change the product assortment. When there is no demand for a product of the product assortment, it will be removed from the assortment **(1.2.2)**. The demand and the mission will lead to the modification of products and the development of new products. The reduced product assortment will be completed with these new and modified products **(1.2.3)**.

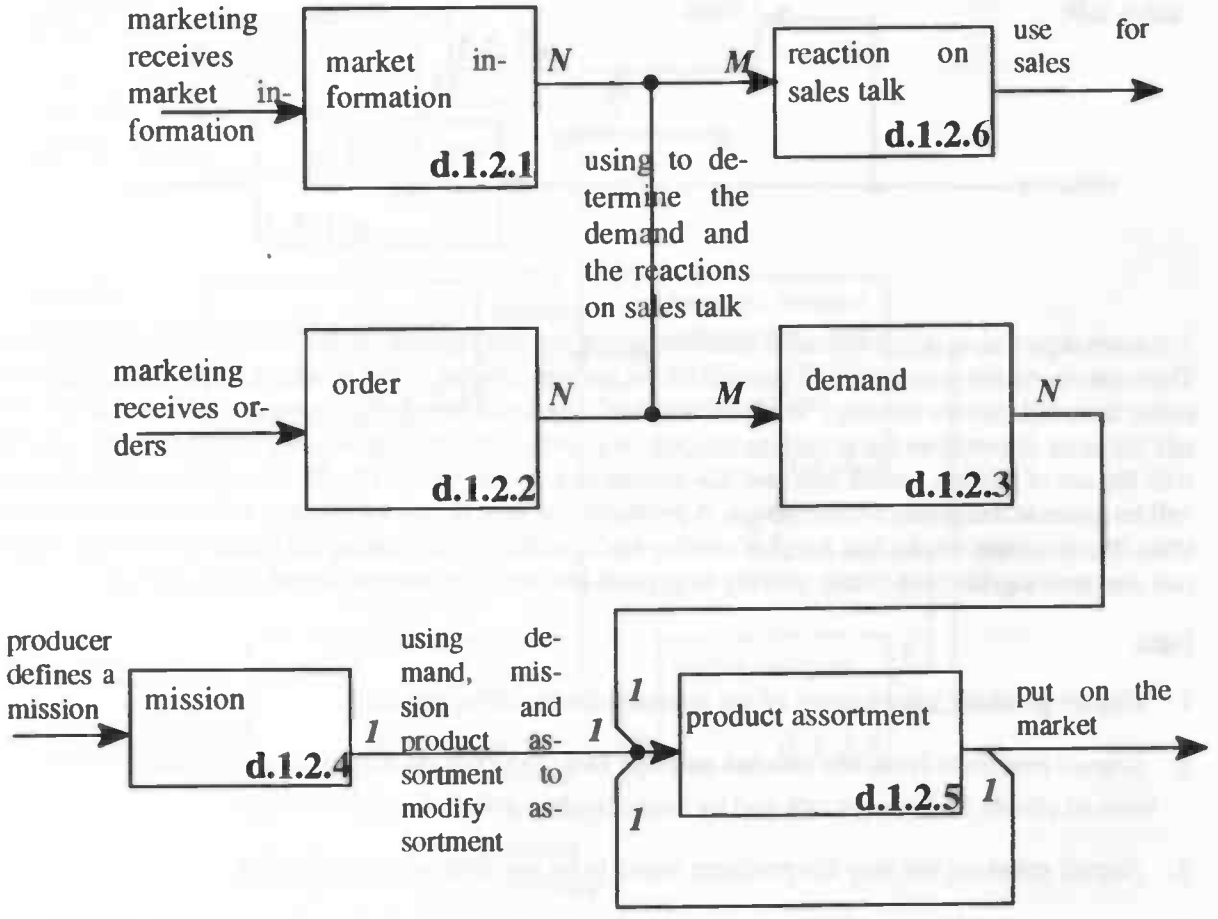
Data:

1. *(input)* **mission:** the way the producer wants to be and behave on the market.
2. *(input)* **orders:** the products bought by a client.
3. *(input)* **market information:** personal and behavioral information about the client.
4. *(input)* **product assortment:** all the products the producer produces.
5. *(internal)* **demand:** the demand for products from the market.
6. *(internal)* **product assortment:** all the products the producer produces.
7. *(output)* **product assortment:** all the products the producer produces.
8. *(output)* **reactions on sales talk:** information acquired by analyzing the reactions of clients on the sales talk and by analyzing the orders of specific clients.

Activities:

1. **analyze market:** analyze all the information received from the market.
2. **stop production of products:** remove products from the product assortment when there is no demand for a specific product or the mission of the producer has been changed.
3. **modify and develop products:** modify products of the product assortment or add new products to the product assortment when the demand and/or the mission are changed.

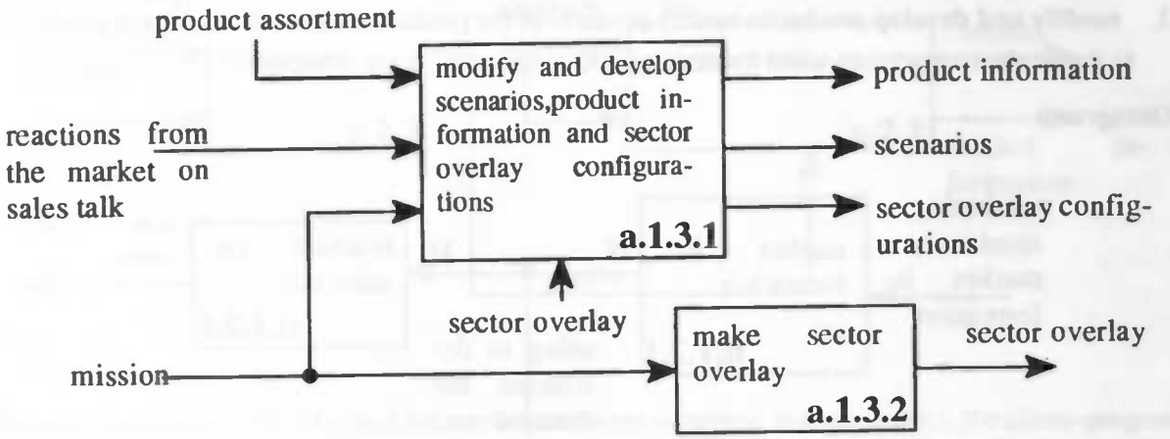
**Datagram:**




The instances of the data types market information **(d.1.2.1)** and order **(d.1.2.2)** are generated by the clients. These instances will be filtered, which results in instances of the data type reactions on sales talk **(d.1.2.6)** and demand **(d.1.2.3)**. The demands will be used to make and/or modify a product assortment **(d.1.2.5)**. The product assortment itself will also be used to compose a new product assortment. The mission **(d.1.2.4)** of the producer also influences the composition of a product assortment.

**level 2: decomposition of *sales (a.1.3)***

**Actigram:**




The sales department develops one or more scenarios for every product of the product assortment **(1.3.1)**. The contents of the scenarios will depend on the current mission of the producer. It also has to fit in the entire sales talk (sector overlay). When clients don't react positive on a certain part (scenario) of the sales talk the sales department has to change this part. By using a scenario a producer can recommend a product with the aid of movies, sound, text and 3D–actions in a virtual world. The 3D description of the products will be given in the product information. A producer can tune the sector overlay with his own configurations. If a producer thinks that a sector overlay isn't capable of expressing his image in the right way, he can also develop his own sector overlay to express his image in his own favorite way **(1.3.2)**.

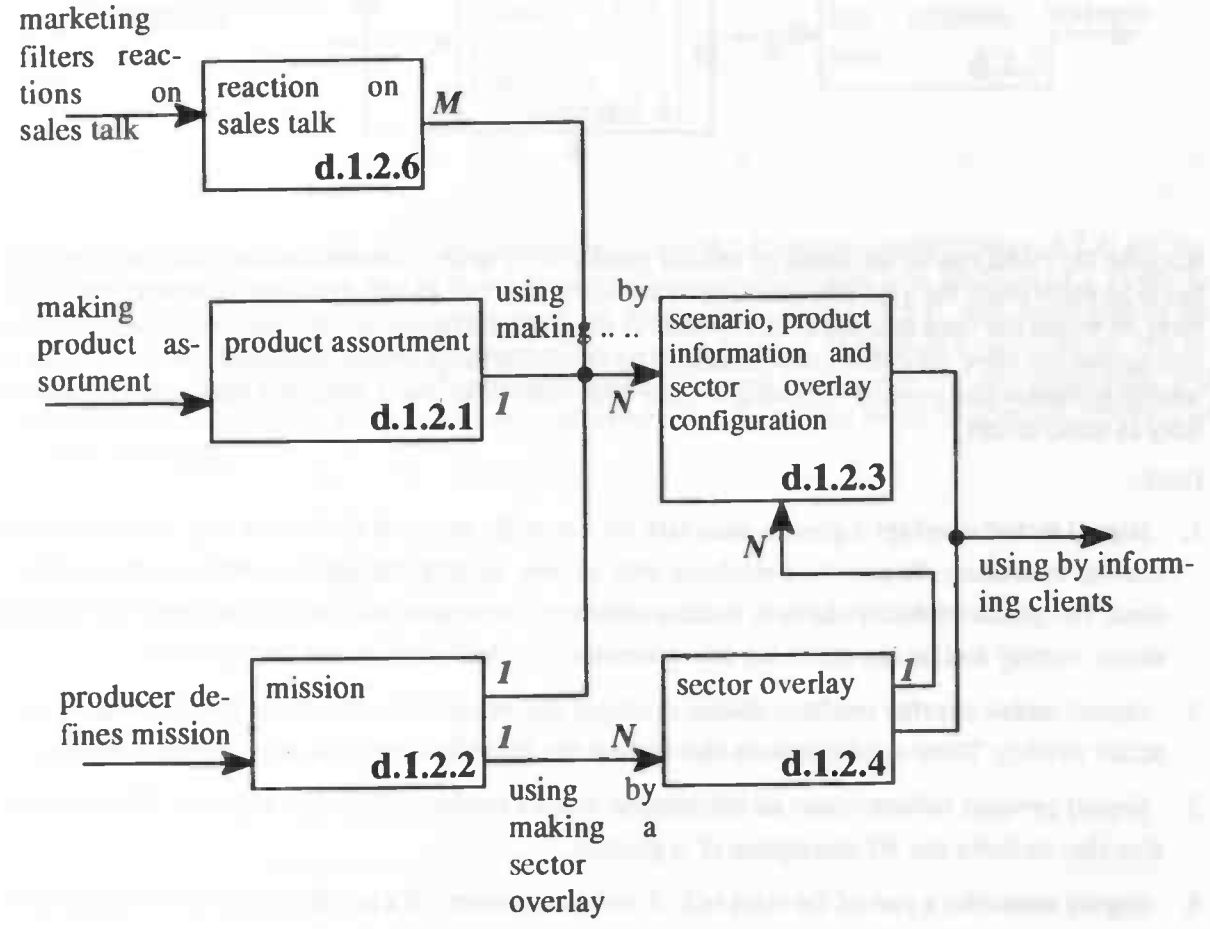
Data:

1. *(input)* **product assortment:** all the products the producer produces.
2. *(input)* **reactions from the market on sales talk:** information acquired by analyzing the reactions of clients on the sales talk and by analyzing the orders of specific clients.
3. *(input)* **mission:** the way the producer wants to be and behave on the market.
4. *(output)* **product information:** all information about a product, like price, name etc. This information also includes the 3D description of a product.
5. *(output)* **scenario:** a part of the sales talk. A scenario consists of a recommendation of a product by using movies, sounds and 3D–actions in a virtual world in the producers favorite way.
6. *(output)* **sector overlay:** a generic sales talk for a specific sector of the market (e.g. domestic architecture, insurance). To provide a producer with an own identity, the sector overlay can be configured. The producer doesn't have to make a completely new sales talk; he can configure the existing sector overlay and he can create his own scenarios in order to express his own identity.
7. *(output)* **sector overlay configurations:** producer dependent information needed to configure the sector overlay. These configurations also include the identification of the needed sector overlay.

Activities:

1. **modify and develop product information, scenarios and sector overlay configurations:** the making of new or changed product information, scenarios and sector overlay configurations, taking the current sector overlay into account.
2. **make sector overlay:** developing a producer–specific sector overlay by using the mission of the producer.

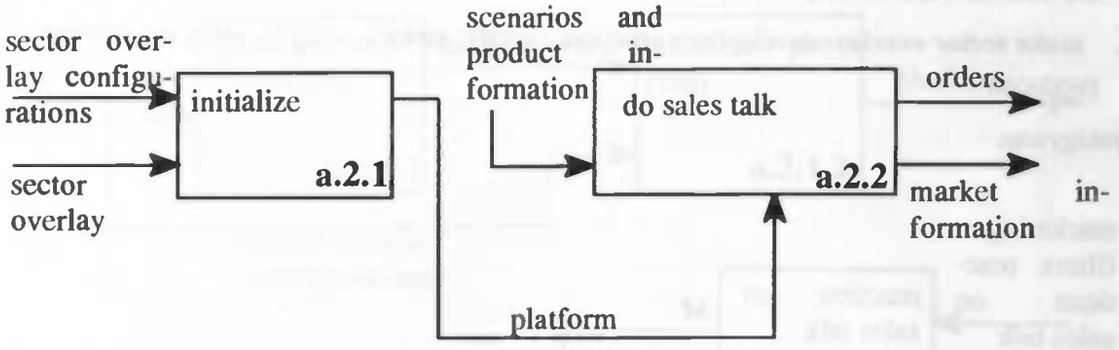
**Datagram:**




The marketing department generates the instances of the data types reaction on the sales talk (**d.1.2.6**) and product assortment (**d.1.2.1**). These instances are used, together with the mission of the producer, to make instances of the separate data types scenario, product information and sector overlay configuration (**d.1.2.3**). These instances will be created for a sector overlay (**d.1.2.4**), which will be made by using the mission of the producer. The scenarios, product information, sector overlay configurations and the sector overlay itself, will eventually be used to inform the clients.

**level 1: decomposition of *inform (a.2)***

**Actigram:**




Because the client can be informed by several producers of several market–sectors, the client–program has to be made ready for a specific producer of a specific sector (2.1). After this initialization step the platform on which the sales talk has to be executed is ready. Furthermore, all information of the sales talk is also accessible. Now the client can be informed by executing the producer dependent sales talk, using the needed scenarios and product information (2.2). This sales talk results in market information and hopefully in some orders.

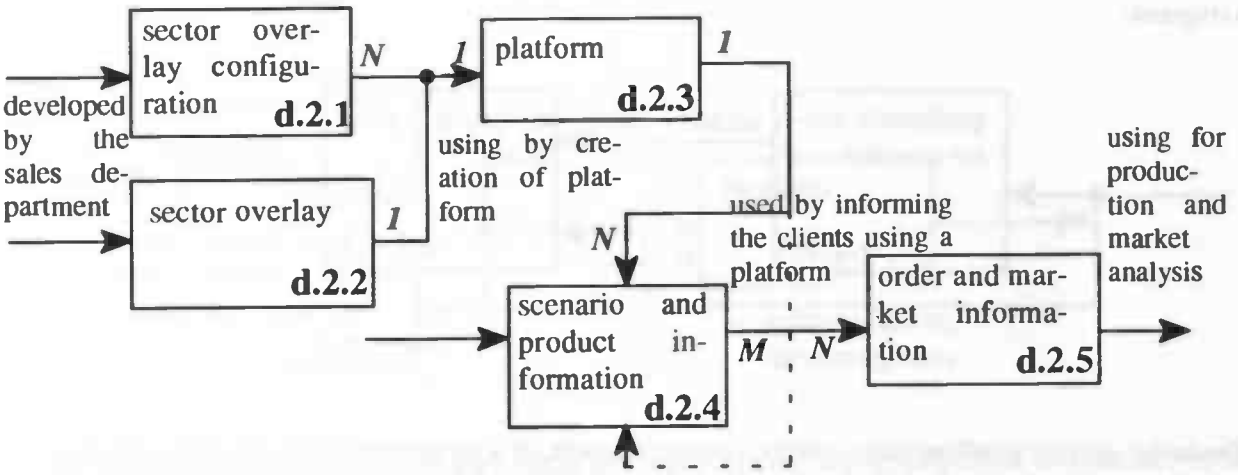
Data:

1. *(input)* **sector overlay:** a generic sales talk for a specific sector of the market (e.g. domestic architecture, insurance). To provide a producer with an own identity, the sector overlay can be configured. The producer doesn't have to make a completely new sales talk; he can configure the existing sector overlay and he can create his own scenarios in order to express his own identity.
2. *(input)* **sector overlay configurations:** producer dependent information needed to configure the sector overlay. These configurations also include the identification of the needed sector overlay.
3. *(input)* **product information:** all information about a product, like price, name etc. This information also includes the 3D description of a product.
4. *(input)* **scenario:** a part of the sales talk. A scenario consists of a product commercial using movies, sounds and 3D–actions in a virtual world in the producers favorite way.
5. *(internal)* **platform:** the producer dependent environment on which the sales talk can be executed.
6. *(output)* **orders:** the products bought by a client.
7. *(output)* **market information:** personal and behavioral information about the client.

Activities:

1. **initialize:** the creation of a producer dependent platform and making the producer dependent sales talk accessible.
2. **do sales talk:** the execution of the sales talk on the producer dependent platform. This execution results in market information and possibly some orders.

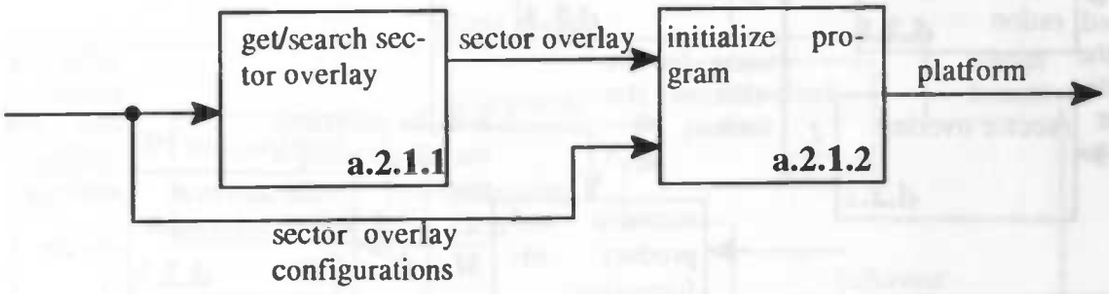
**Datagram:**




The instances of the data types sector overlay configuration **(d.2.1)** and the sector overlay **(d.2.2)** are provided by the sales department. Out of these instances a instance of the data type platform **(d.2.3)** will be created. With instances of the data types scenario en product information **(d.2.4)**, also developed by the sales department, the client will be informed while using an instance of the data type platform. The informing of the client will often result in the generation of several instances of the data types order and market information **(d.2.5)**. These latter instances will eventually be used for the production process and for market analysis.

**level 2: decomposition of *initialize (a.2.1)***

**Actigram:**




The sector overlay configurations consists, amongst others, of a sector overlay id. By using this id the correct sector overlay can be found or downloaded if necessary (**a.2.1.1**). To save time a lot of sector overlays will be stored in a special sector overlay database at the workspace of the client. If the sector overlay can't be found in this database, it has to be downloaded from the supplier. The program will be initialized when the sector overlay is available. By using the sector overlay configurations, the supplier can give the resulting platform his own identity.

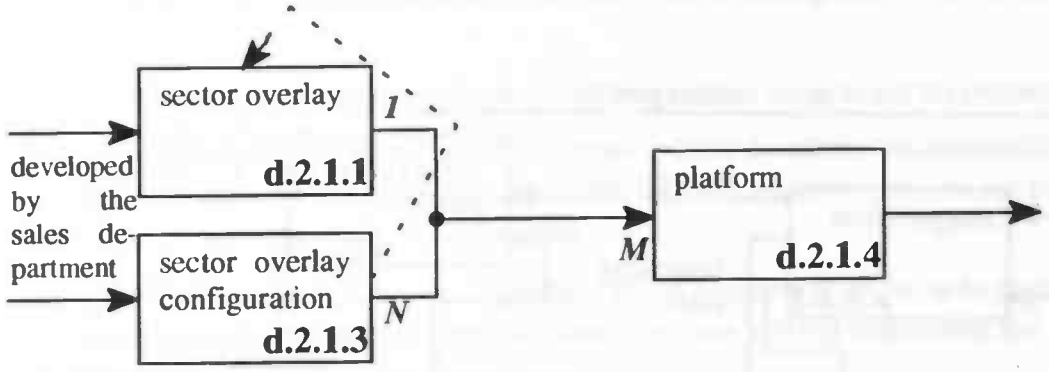
Data:

1. *(input)* **sector overlay configurations:** producer dependent information needed to configure the sector overlay. These configurations also include the identification of the needed sector overlay.
2. *(internal)* **sector overlay:** a generic sales talk for a specific sector of the market (e.g. domestic architecture, insurance). To provide a producer with an own identity, the sector overlay can be configured. The producer doesn't have to make a completely new sales talk; he can configure the existing sector overlay and he can create his own scenarios in order to express his own identity.
3. *(output)* **platform:** the producer dependent environment on which the sales talk can be executed.

Activities:

1. **get/search sector overlay:** the sector overlay has to be found in the sector overlay database of the client or at the producers database. When the sector overlay has been found the initialization can start.
2. **initialize program:** the user program has to be made ready for a specific sector and supplier.

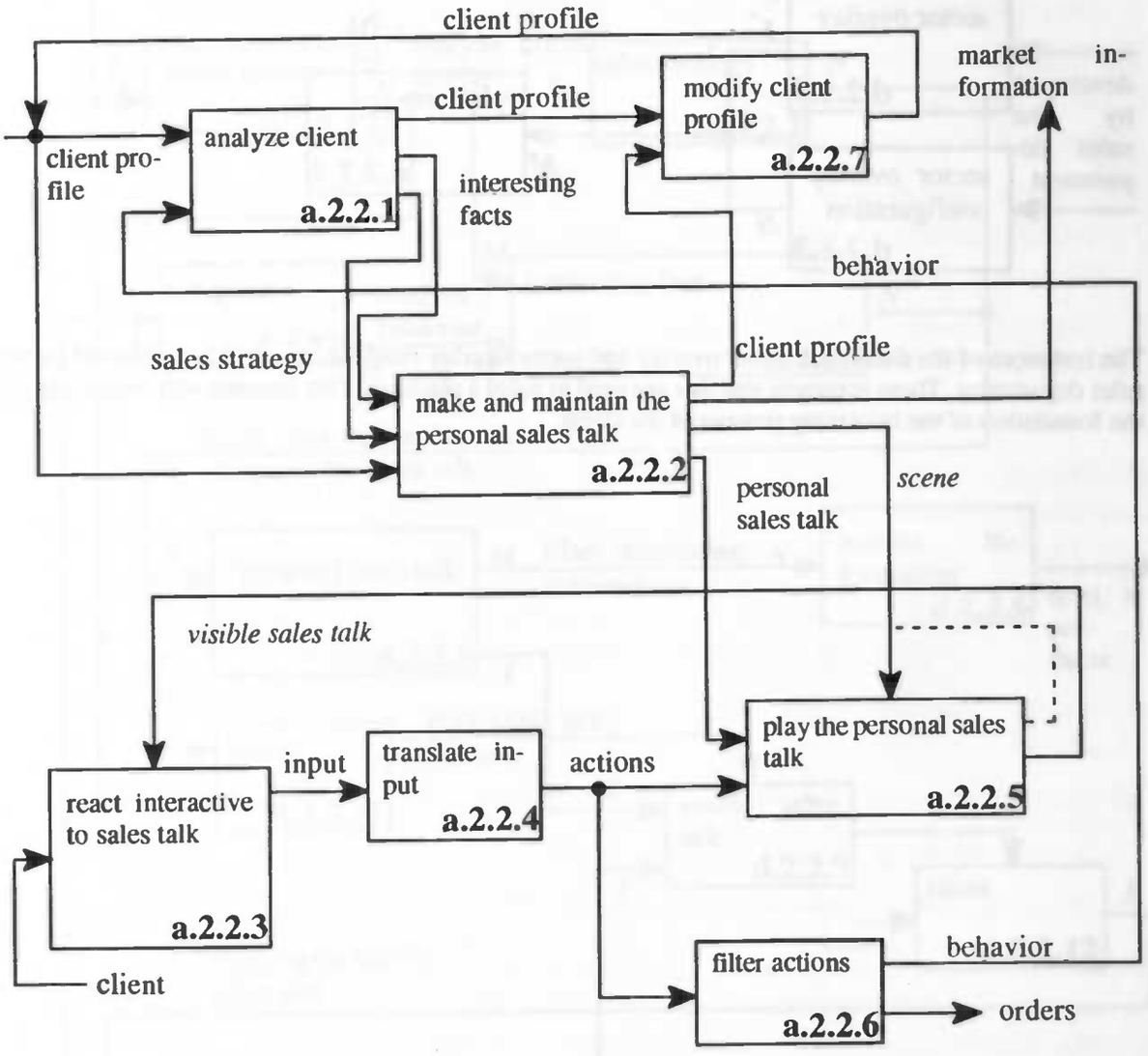
**Datagram:**




The instances of the data types sector overlay and sector overlay configuration will be generated by the sales department. These instances together are used to build a platform. This instance will eventually lay the foundation of the informing process of the client.

**level 2: decomposition of *do sales talk (a.2.2)***

**Actigram:**




The central activity of this decomposition is the making and maintaining of the personal sales talk (2.2.2). As noticed the sales talk has been complemented with the adjective 'personal'. This means that the generic sales talk, which was producer dependent, at this moment has been adapted to the specific client. The making and maintaining of this personal sales talk will be done by using the results of the client analysis (2.2.1). This analysis results in a sales strategy and some vague interesting facts. This last term is vague, because each sector has its own interesting facts that could occur during a sales talk. The client profile also affects the making and maintaining of the personal sales talk. The result of this making and maintaining is of course the personal sales talk itself, but the market information can also be filtered during this activity. This process also causes changes in the 3D virtual world (also called scene). Another side–effect can be changes in the client profile, which will be processed in the modification of the client profile (2.2.7).

The client analysis consists of a person analysis and a behavior analysis. The first will be done by means of a questionnaire, the second analysis will need behavior data to analyze. So while playing the personal sales talk (2.2.5) the client reactions have to be captured. This can be done because all the interactive client

reactions to the visible sales talk (2.2.3) will result in input. This input of the client will be translated to actions (2.2.4). These actions also affect the visible sales talk. The actions can be filtered (2.2.6) to the needed behavior. The filtering of the actions can also result in sending orders to the producer.

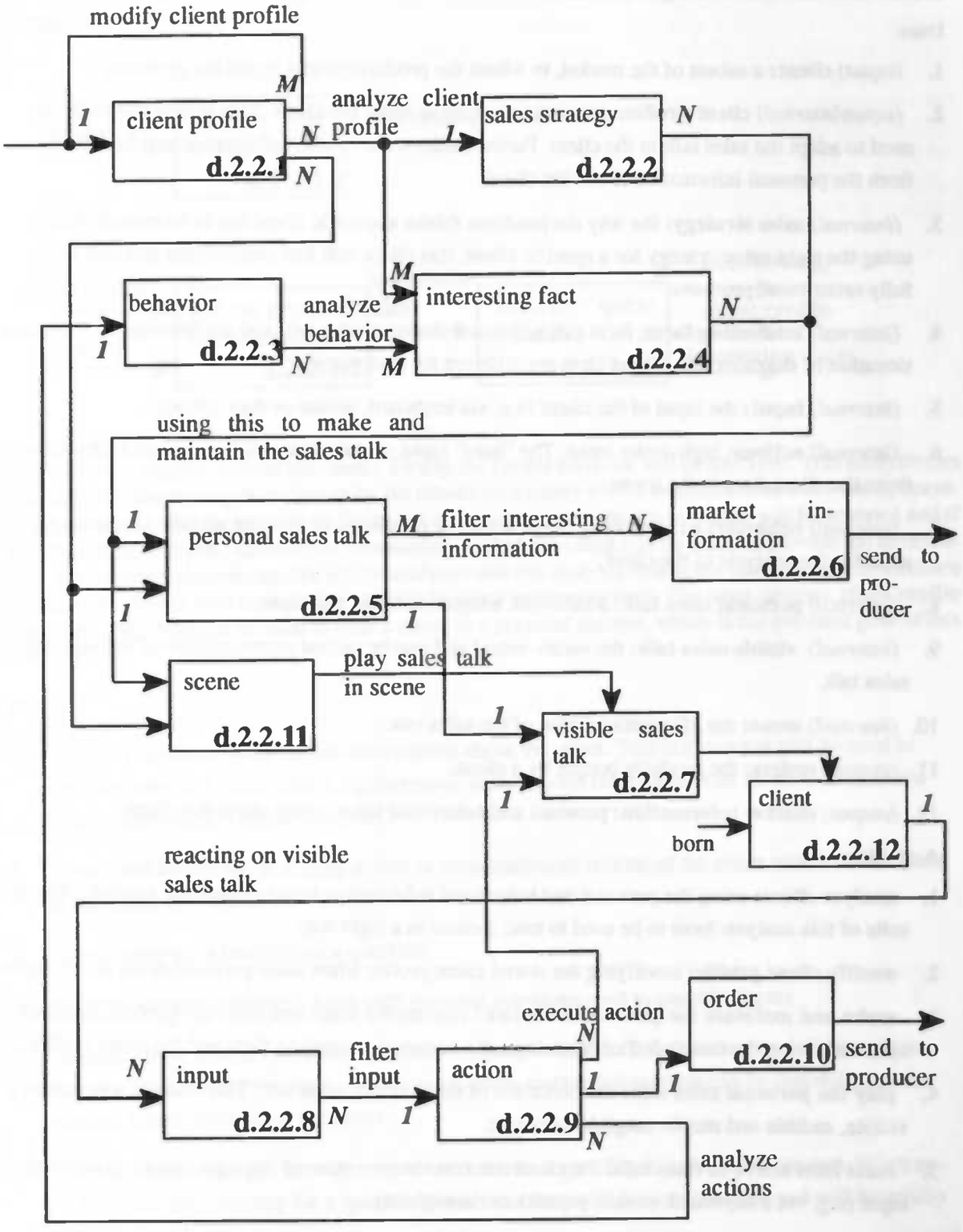
Data:

1. *(input)* **client:** a subset of the market, to whom the producer wants to sell his products.
2. *(input/internal)* **client profile:** personal information about the client. This information will be used to adapt the sales talk to the client. Furthermore, some market information can be derived from the personal information about the client.
3. *(internal)* **sales strategy:** the way the producer thinks a specific client has to be treated. When using the right sales strategy for a specific client, this client will feel comfortable and will hopefully order more products.
4. *(internal)* **interesting facts:** facts that will occur during a sales talk and are important for the continuation of the sales talk. These facts are different for each sector.
5. *(internal)* **input:** the input of the client (e.g. via keyboard, mouse or data–glove).
6. *(internal)* **actions:** high–order input. The "hard" input of the user will be translated into the actions the client does in the scene.
7. *(internal)* **behavior:** an interesting client action or combination of client actions which can be used for the analysis of the client.
8. *(internal)* **personal sales talk:** a sales talk adapted to a specific client.
9. *(internal)* **visible sales talk:** the audio–visual and maybe tactual representation of the personal sales talk.
10. *(internal)* **scene:** the 3D representation of the sales talk.
11. *(output)* **orders:** the products bought by a client.
12. *(output)* **market information:** personal and behavioral information about the client.

Activities:

1. **analyze client:** using the personal and behavioral information about a client for analysis. The results of this analysis have to be used to treat a client in a right way.
2. **modify client profile:** modifying the stored client profile when some personal items are changed
3. **make and maintain the personal sales talk:** continually make and color the generic, standard sales talk using the concluded sales strategy, the occurred interesting facts and the client profile.
4. **play the personal sales talk:** the execution of the personal sales talk. This execution results in a visible, audible and maybe tangible sales talk.
5. **react interactive to sales talk:** the client react on the execution of the sales talk by generating input (e.g. via a keyboard, mouse, joystick or data–glove).
6. **translate input:** the input of the client will be translated into a high–order input (actions).
7. **filter actions:** the actions will be filtered to behavior or orders.

**Datagram:**

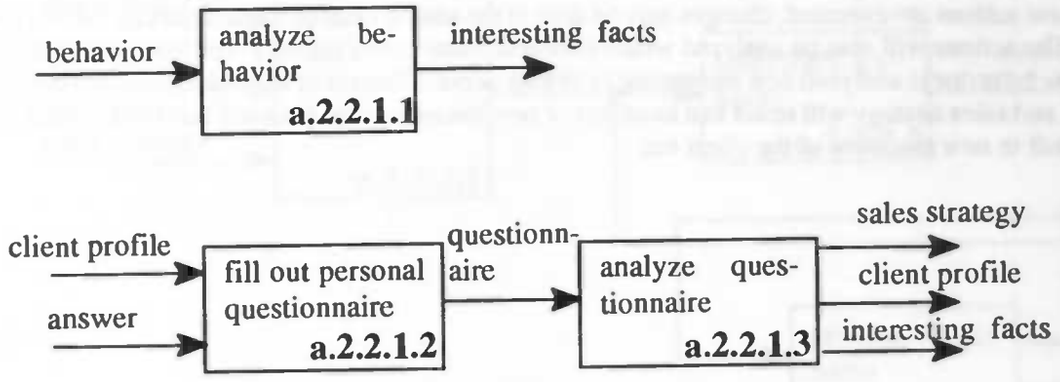



The instance of the data type client profile (**d.2.2.1**) will grow and change during the life–time of ITS MAGIC. The client profile will also be used to determine instances of the data types sales strategy (**d.2.2.2**) and interesting facts (**d.2.2.4**). By using these instances and the client profile, an instance of the

data type personal sales talk can be made **(d.2.2.5)** as well as an instance of the data type scene **(d.2.2.11)**. When playing the personal sales talk in the scene an instance of the data type visible sales talk **(d.2.2.7)** will be generated. It is also possible to filter interesting information from the personal sales talk, which results in instances of the data type market information **(d.2.2.6)**. This visible sales talk can be observed by a client **(d.2.2.12)**. When the client reacts on this sales talk, instances of the data type input **(d.2.2.8)** will be generated. This input will be filtered which results in instances of the data type action **(d.2.2.9)**. When these actions are executed, changes may be seen in the visible sales talk and orders **(d.2.2.10)** may appear. The actions will also be analyzed which results in instances of the data type behavior **(d.2.2.3)**. When this behavior is analyzed new interesting facts may occur. Changes of the instances of the interesting facts and sales strategy will result in a modified or new instance of the personal sales talk, which will often result in new reactions of the client etc.

**level 3: decomposition of *analyze client (a.2.2.1)***

**Actigram:**




There are two ways to analyze the clients. Firstly, the clients behavior will be analyzed. This analysis can result in some interesting facts. Secondly, the clients personality will be analyzed. This will from a personal questionnaire. For the process of filling out the questionnaire, the client profile will be scanned and if the client profile doesn't contain the information needed, the client will be asked to provide the information. The completed questionnaire will be analyzed and this analysis results in a sales strategy, sometimes in a modification of the client profile and maybe some interesting facts. The sales strategy, client profile and interesting facts can be used to treat a client in a personal manner, which is the eventual goal of this analysis.

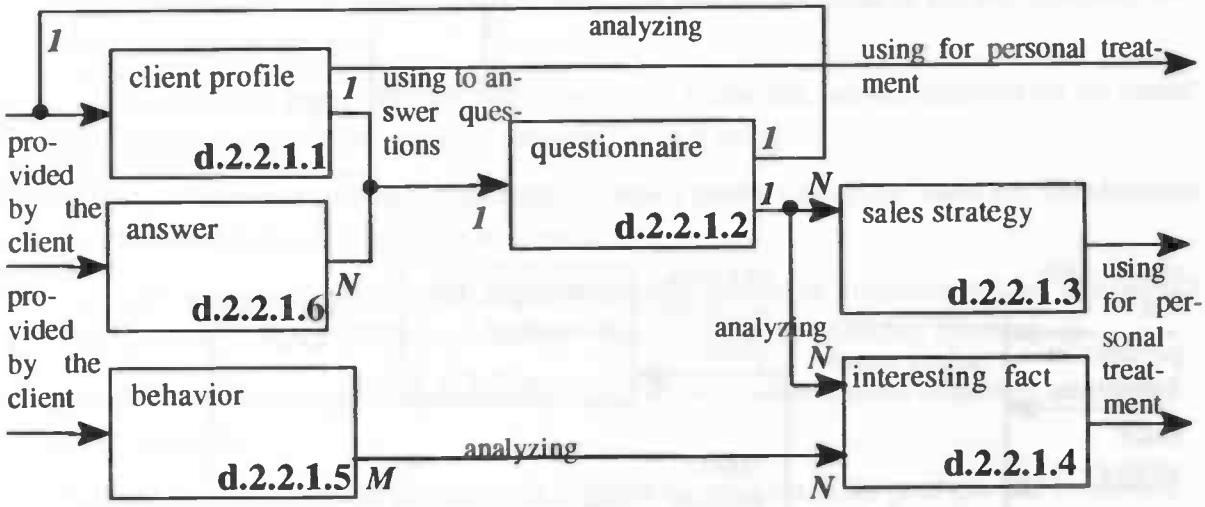
Data:

1. *(input)* **client profile:** personal information about the client. This information will be used to adapt the sales talk to the client. Furthermore, some market information can be derived from the personal information about the client.
2. *(input)* **behavior:** an interesting action or combination of actions of the client which can be used for the analysis of the client.
3. *(input)* **answer:** a reaction on a question.
4. *(internal)* **questionnaire:** a form with personal questions used to classify clients.
5. *(output)* **client profile:** personal information about the client. This information will be used to adapt the sales talk to the client. Furthermore, some market information can be derived from the personal information about the client.
6. *(output)* **sales strategy:** the way the producer thinks a specific client has to be treated. When using the right sales strategy for a specific client, this client will feel comfortable and will hopefully order more products.
7. *(output)* **interesting facts:** facts that will occur during a sales talk and are important for the continuation of the sales talk. These facts are different for each sector.

Activities:

1. **analyze behavior:** analyzing the actions, which the client performs during a sales talk, to determine interesting facts, which can be used to treat the client in a personal manner.
2. **fill out personal questionnaire:** filling out a form by using the information in the client profile. When more information is needed, the client will be asked to provide this information.
3. **analyze questionnaire:** analyzing the completed form, which will result in a classification of the client (sales strategy). This analysis sometimes results in changes in the client profile. Also some interesting facts can occur, which can be used to treat the client even more personal.

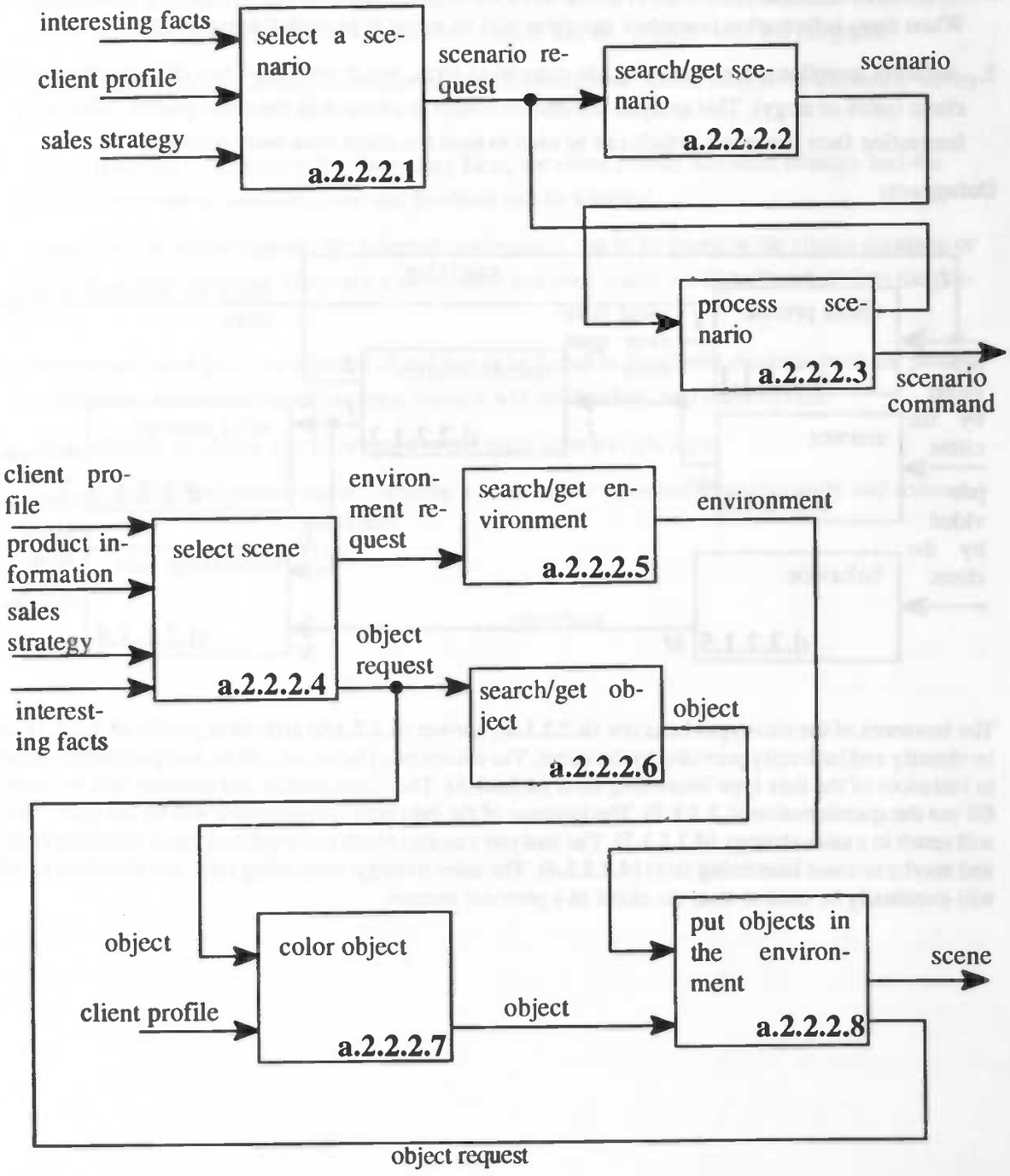
**Datagram:**




The instances of the data types behavior (**d.2.2.1.5**), answer (**d.2.2.1.6**) and client profile (**d.2.2.1.1**) will be directly and indirectly provided by the client. The instances of behavior will be analyzed which results in instances of the data type interesting facts (**d.2.2.1.5**). The client profile and answers will be used to fill out the questionnaire (**d.2.2.1.2**). The instance of the data type questionnaire will be analyzed which will result in a sales strategy (**d.2.2.1.3**). The analysis can also result in a modification of the client profile and maybe in some interesting facts (**d.2.2.1.4**). The sales strategy, interesting facts and the client profile will eventually be used to treat the client in a personal manner.

**level 3: decomposition of *make and maintain the personal sales talk (a.2.2.2)***

**Actigram:**




By analyzing the interesting facts, client profile and sales strategy, the next scenario of the sales talk will be selected. The requested scenario has to be downloaded from the producer. If the client has contacted the provider before, the scenario may be found at the client's workspace (2.2.2.2). After downloading the scenario it will be processed, which possibly results in the execution of several scenario commands

**(2.2.2.3)**. The scenario commands have to be executed one by one. Like the scenarios, a scene also has to be selected **(2.2.2.4)**. A scene consists of an environment with several (3D) objects. These objects and the environment also have to be downloaded or fetched from the clients workspace **(2.2.2.5 and 2.2.2.6)**. The generic downloaded objects have to be adapted to the client. So by using information of the client profile the objects will be "colored". After this coloring the objects have to be placed at the right position in the environment, after which the scene is created **(2.2.2.8)**.

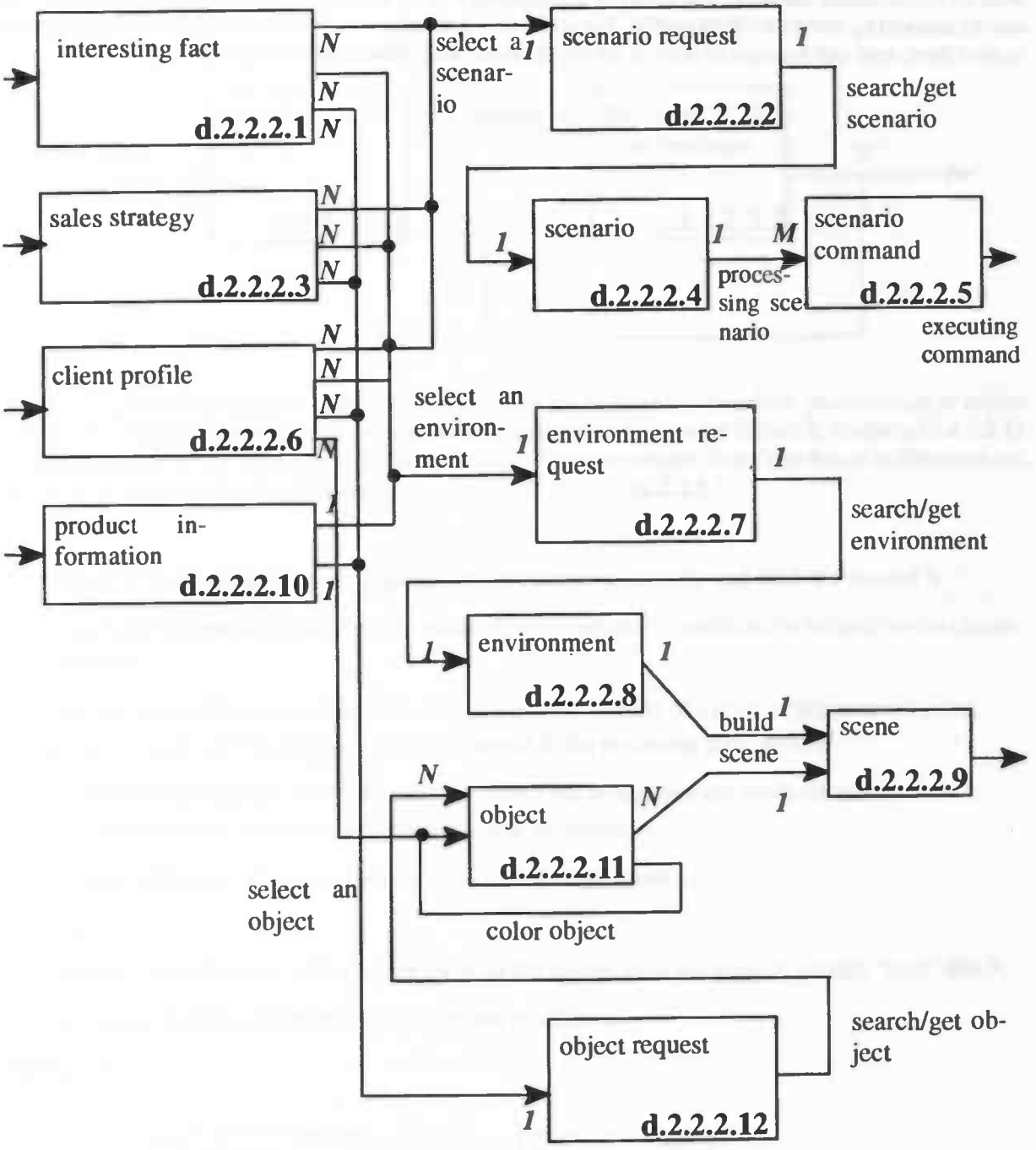
Data:

1. *(input)* **client profile:** personal information about the client. This information will be used to adapt the sales talk to the client. Also some market information can be filtered from the personal information about the client.
2. *(input)* **sales strategy:** the way the producer thinks a specific client has to be treated. When using the right sales strategy for a specific client, this client will feel comfortable and will hopefully order more products.
3. *(input)* **interesting facts:** facts that will occur during a sales talk and are important for the continuation of the sales talk. These facts are different for each sector.
4. *(input)* **product information:** all information about a product, like price, name etc. This information also includes the 3D description of a product.
5. *(internal)* **scenario:** a part of the sales talk. A scenario consists of a recommendation of a product by using movies, sounds and 3D–actions in a virtual world in the producers favorite way.
6. *(internal)* **scenario request:** this request contains all information needed to identify and fetch a specific scenario.
7. *(internal)* **environment:** the environment in which the products of the producer will be shown. (e.g. the house of a client in which a kitchen will be placed).
8. *(internal)* **environment request:** this request contains all information needed to identify the required environment.
9. *(internal)* **object:** A 3D representation of a product of the producer.
10. *(internal)* **object request:** this request contains all information needed to identify and fetch an object.
11. *(output)* **scenario command:** A scenario consists of several actions which have to be split in separate commands, in order to be able to execute the scenario in the scene. Such an action is called a scenario command.
12. *(output)* **scene:** the 3D representation of the sales talk.

Activities:

1. **select a scenario:** by analyzing the interesting facts, the client profile and the sales strategy, a next, suitable scenario can be selected.
2. **search/get a scenario:** the selected scenario has to be found in the clients database or in the producers database. When the scenario has been found it will be "loaded" into the program.
3. **process a scenario:** a scenario consists of several commands, which have to be executed one by one.
4. **select a scene:** by analyzing the interesting facts, the client profile, the sales strategy and the product information, suitable scene and products can be selected.
5. **search/get an environment:** the selected environment has to be found in the clients database or in the producers database. When the environment has been found it will be "loaded" into the program.
6. **search/get an object:** the selected object has to be found in the clients database or in the producers database. When the object has been found it will be "loaded" into the program.
7. **color object:** an object will be adapted to the users taste and life style.
8. **put objects in the environment:** creating a scene out of the selected environment and colored objects.

**Datagram:**

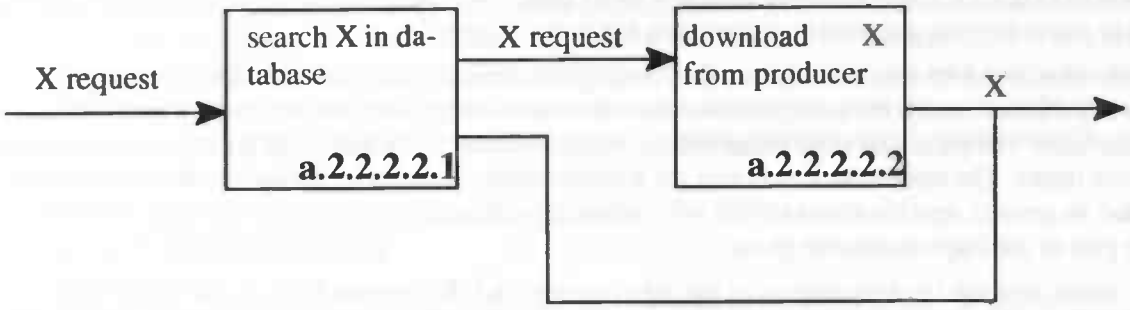



By using instances of the data types interesting fact (**d.2.2.2.1**), sales strategy (**d.2.2.2.3**) and client profile (**d.2.2.2.6**), a scenario can be selected. This results in an instance of the data type scenario request (**d.2.2.2.2**). When this request is processed, an instance of the data type scenario (**d.2.2.2.4**) will be created. This scenario will be processed into several instances of the data type scenario command (**d.2.2.2.5**). By using instances of the three first named data types and instances of the data type product information (**d.2.2.2.10**), an environment can be selected, which results in an instance of environment request (**d.2.2.2.7**). The processing of this request will result in an environment instance (**d.2.2.2.8**). Instances of these four data types can also be used to determine the objects to be used. This results in an object re-

quest instance **(d.2.2.2.12)**. This request will be processed into an object instance, which can be combined with an environment instance. The result of this process will be a scene instance. An object instance can also be colored by using the client profile. The scene instance and scenario command instances are adapted to the client, and will be used to inform the client about the products of the supplier.

**level 4: decomposition of *search/get X {X = scenario | environment | object} (2.2.2.2, 2.2.2.5 and 2.2.2.6)***

**Actigram:**




In this actigram three actigrams are taken together. For X can be read *scenario* or *environment* or *object*. While processing a X request, the first step will be a search in the special clients X database **(2.2.2.2.1)**. In this database, X's of several producers are stored. When the needed X can't be found in this database, it has to be downloaded from the currently contacted producer **(2.2.2.2.2)**.

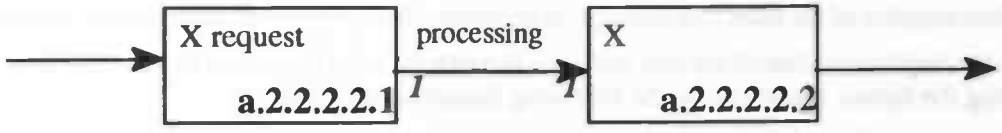
Data:

1. *(input)* ***X* request:** this request contains all information to identify and fetch the needed X.
2. *(internal)* **X request:** same request, issued when the needed X could not be located on the clients computer
3. *(output)* **scenario:** a part of the sales talk. A scenario consists of a product presentation using movies, sounds and 3D–actions in a virtual world in the producers favorite way.
4. *(output)* **environment:** the environment in which the products of the producer will be shown. (e.g. the house of a client in which a kitchen will be placed)
5. *(output)* **object:** A 3D representation of a product of the producer.

Activities:

1. **search X in database:** checking for the X in the special local database. If present "load" this X.
2. **download X from producer:** get X from the producer

**Datagram:**



## 3.5 Conclusions

In this section some conclusions will be drawn from the formal specification. Firstly, we discovered, during the specification process, that **analyze market (a.1.2.1)** is almost the same kind of process as **analyzing client (a.2.2.1)**. Both processes receive a lot of data. These data have to be analyzed and the results have to show the way the market or the client has to be "treated".

Besides that, it can be seen that the way the clients get informed influences the **sales** process **(a.1.3)**. The sales department creates the configurations and other kinds of information which are needed to initialize the platform. The parts that need initialization, mainly located in **do sales talk (a.2.2)**, are not specified in more detail. The reason for this is that for a more detailed specification, implementation aspects are needed. In general, specifications should not contain implementation aspects, because they are considered to be part of the implementation phase.

This means that the implementation of the sales process **(a.1.3)** depends fully on the implementation of the **do sales talk** process **(a.2.2)**. The sales process will eventually contain a lot of intelligent tools to help managers with making and modifying the sales talk (see section 8.4.3). The process **produce physically (a.1.1)** falls outside the scope of ITS MAGIC and isn't specified in more detail.

So, in summary, a part of **produce (a.1)**, namely **(a.1.2.1)**, can be developed by reusing parts of **(a.2)**. Another part, **(a.1.3)**, is dependent on more detailed information of the development of **(a.2)**. And **(a.1.1)** is not interesting for ITS MAGIC. Therefore, we think that the development of the **(a.2)** part of the formal specification is more important and interesting than developing the **(a.1)** part. From now on, we will call the **(a.2)** part the user part of the system and the **(a.1)** part the supplier part. Furthermore the user part is more interesting because this is the part of the system the clients have to work with. The supplier will be more interested in how the market (the clients) react to the system, than how the persons, who have to take care of making a sales talk, react to the system. The making of the sales talk eventually has to be as easy as possible, but can be done by hand at this moment. In the future, a lot of intelligent tools have to be developed, to support the managers in making and maintaining their sales talks. These managers should be able to do this job without any knowledge of the technical aspects of the system. When developing the supplier part, reuse of software is also an important issue. By setting the correct configurations, the software should be adaptable to the particular situation of a supplier.

The user part of ITS MAGIC also seems to be the most complicated part of the system, because this part has to interact "live" (real–time) with the client. In contrast, the processes of the supplier part are off–line processes. So, to show the most interesting functionality of ITS MAGIC, we will implement the user part of the system.

Reading the previous summary, it seems obvious that we decided to develop and implement the user part of ITS MAGIC, because:

- the user part is the most complicated process, because of real–time constraints.
- the user part is the most interesting process (to suppliers and ourselves).
- reuse of software possible (e.g. to build the marketing part of the supplier part).
- the implementation of the sales part depends on decisions during the development of the user part.

The first step to the implementation of the user part is to identify the most important elements of the system. By analyzing the formal specification, the following elements can be found:

- the initialization of the user part (a.2.1)
- virtual salesman; the making and maintaining of the sales talk (a.2.2.2)
- the personal and behavioral analysis of the client (a.2.2.1)
- the processing of the user input to actions in the visible sales talk (a.2.2.3, a.2.2.4 and a.2.2.5)
- administrations are needed to store scenarios (a.2.2.2.2), environments (a.2.2.2.5), objects (a.2.2.2.6) and sector overlays (a.2.1.1)

Finally, we think that this chapter describes our ideas in a formal, clearly structured manner. During the specification process the ideas were adjusted and improved. At the beginning of this process the ideas "lived their own lives", and at the end they were united into one. This result helped us with the developing and implementing of the user part. During the implementation no unexpected changes of ideas occur, because the main ideas were settled. Minor adjustments or even new ideas occurred in the lower levels, but they didn't affect the specified ideas. During every specification process, we have to decide at what level we want to stop specifying. Our motivation to stop specifying at the level we described in this chapter, was to leave the developer of a particular part of the system free to develop his own ideas. He just had to keep the interfaces, provided by the specification, in mind. In this way, the system could be developed and implemented in a parallel manner, because the specification provided interfaces between the important elements of the system.

## Chapter References


[1] Douglas T. Ross, "Structured Analysis (SA): A Language for Communicating Ideas", *IEEE Transactions on Software Engineering*, Vol. SE−3, No. 1, pp 16 − 34, January 1977.

[2] Douglas T. Ross and Kenneth E. Schomanm, Jr., "Structured Analysis for Requirements Definition", *IEEE Transactions on Software Engineering*, Vol. SE−3, No. 1, pp 6 − 15, January 1977.

[3] J.D. Aron, *The Program Development Process; The programming team part II*, pp 276 − 270.

[4] Prof. Dr. J.C. van Vliet, *Software Engineering*, pp 71 − 74.

[5] "Basic Reference Model of Open Distributed Processing, part 1, overview and guide to use", *ITU/T X.901−ISO N7053−1*, November 1993.

# Chapter 4 Implementation Aspects of ITS MAGIC

## 4.1 Introduction

In the previous chapter we have given a specification of the ITS MAGIC system. In this chapter, we will describe how part of the system will be implemented. We use the specification to determine the elements we need, and provide an implementation for most of these elements. We will explain how the program will be implemented and why it will be done in that way. The part of the system we're going to implement is the user part, called the K3 User Program, which is the piece of software a user needs on his computer to be able to work with the system. We will see that certain demands which are not part of the specification play an important role in the architectural decisions which have to be made.

You are going to see the term 'K3' very often throughout this document, so we'll explain this term right now. K3 stands for Kwik, Kwek and Kwak, which are the nicknames given to the authors of this report by the staff members of the Computing Science department of the University of Groningen. In the true Computing Science fashion, these names were abbreviated to KKK, which of course was still too long, so it became K3. Dutch readers will understand that Kwik, Kwek and Kwak are Donald Duck's nephews. In English they are called Huey, Dewey and Louie. Since the authors of this report are not related, we don't think that our nicknames are very well chosen, but it seems impossible to stop people from using them. Lastly, we surrendered and started using 'K3' ourselves.

## 4.2 General Implementation Aspects

Since one of the demands on the system is platform–independence, we can't just write a program for a specific platform. In section 4.3 we will see how we can implement our program in such a way that the platform dependencies are in very specific parts of the program. In this way, only a few files have to be modified to support a new platform.

For the development of a demonstration program a specific target platform is needed. Taking into account the demands on the user's computer hardware set forth in chapter 1, we decided to use the following configuration for the development of our demonstration program.

- Processor: Intel Pentium 150 MHz on a PCI mainboard.
- Memory: 32 Mb.
- Graphics card: Matrox Millennium (with 3D and windows accelerations).
- Operating System: Windows 95.
- Compiler: Borland C++ 4.5 for Windows (of which the ++ part was not used).
- 3D rendering engine: RenderWare DLL version 1.35 DLL Early Access (DEA).

Objections about this configuration were often heard during the development. People tend to think that it is an unrealistic configuration for a home computer. But since the system we're developing will not be in actual use for some years (first the suppliers have to get ready for it, as is explained in chapter 1), it seems very likely that people will actually have the needed processing power by the time our system will be operational.

We decided to use a Borland compiler because it has a very good integrated development environment. Another reason for using a Borland compiler was the presence of a campus licence for it. It is said that the Watcom and Microsoft compilers generate better code, but we did not test them on our program. The RenderWare 3D rendering engine, made by Criterion Software, provided us with a very powerful rendering tool.

## 4.3 MAGIC Architecture

In the initial 'brainstorm phase' of the ITS MAGIC project, we placed several demands on the system. These demands, which together should guarantee the usability for many purposes and the platform independence of the entire system, can be found in chapter 1, but they will be summarized in section 4.3.1. Regardless of the flexibility that is needed, several assumptions about the platform have to be made, because any real system uses a base on which it is built. These assumptions are described in section 4.3.2.

### 4.3.1 MAGIC architectural demands

First of all, the architecture should be open, with which we mean that it should be possible to replace any low–level driver with a newer version or even with a driver from a different vendor. The RenderWare 3D rendering engine which is used in the demonstration program, for example, should be replaceable with any other rendering engine without having to modify the whole system. This means that the K3 User Program should provide the suppliers with a stable environment (or virtual machine if you like), independent of the underlying software.

Secondly, it should be easy for the suppliers to change the information they would like to present to the user. As we have seen in chapter 1, this demand favours the use of a network. The K3 User Program, then, is going to be a networking application which should be able to communicate with any supplier.

Thirdly, the supplier should not have to write his own applications. Rather, he should have the possibility to use a generic program for all suppliers in a certain sector. By configuring this generic program, he should be able to demonstrate his own products to the user in his own way.

Another demand lies in the necessity for the supplier to adapt his approach to a specific client. To facilitate this, it should be possible for the supplier to have the answers to questions in a questionnaire and automatically draw conclusions (i.e. determine a sales strategy which fits the specific client) from these answers. This process is known as Intelligent Personality Identification.

Lastly, it should be possible for the supplier to use and update his client database without having to do any programming. The client database typically contains client characteristics, which are used to place the clients into certain groups. When the marketing department needs to distinguish between persons within an existing group, they should not have to make a lot of changes in some huge rule–base. Rather, they should have the possibility to enter some characteristics for the new group, after which the system should perform all necessary updates.

### 4.3.2 Platform assumptions

The K3 User Program assumes that a windowing system is present. However, one could write a very simple windowing system to implement the few calls that need it. Another assumption that is made is the presence of a timer. The timer is needed for various purposes, as will be explained in the remainder of this chapter and in chapter 5. Yet another assumption is the presence of a C compiler on the target platform.

Of course, any program could be rewritten in another language, but it should be possible to use the program we created when the platform demands are met.

### 4.3.3 The MAGIC System Architecture

Now that we have seen the architectural demands, we can present a system which can meet these demands. The basics of the MAGIC system architecture are shown in Figure 4.1. As can be seen, the architecture is organized as a hierarchy of layers, each one constructed upon the one below it. The bottom three layers together form the K3 User Program.

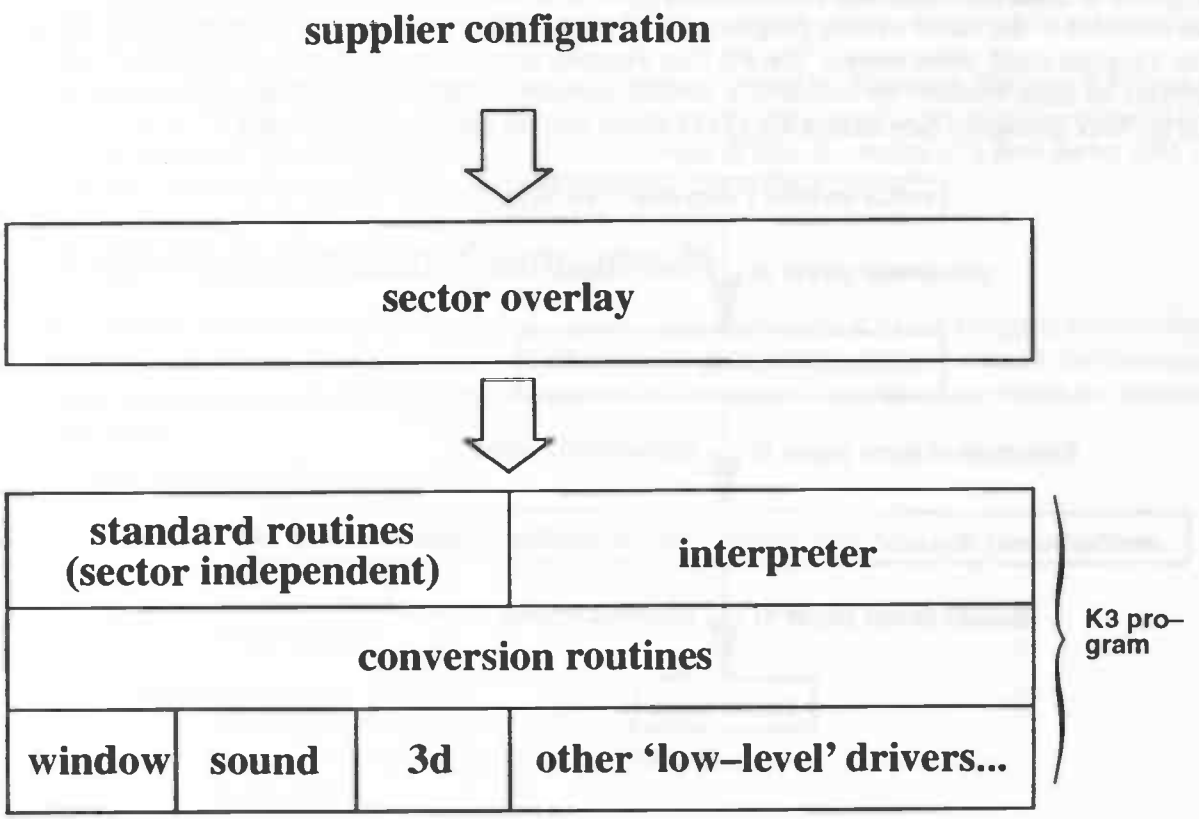



**Figure 4.1:** *The basics of the MAGIC architecture*

The purpose of the lowest layer is to provide the K3 User Program with the necessary functionality. In this layer, for example, the RenderWare 3D rendering engine is placed. The routines RenderWare offers are used to create the 3D routines for the last layer of the K3 User Program (the third layer of the entire system). The latter routines should be stable, i.e. they should not change when a different 3D rendering engine has to be used. To ensure this, the conversion layer is introduced. This layer converts the routines offered into the routines needed. This does not have to be a 1:1 mapping: several low–level routines can be used to create one conversion routine. Together, the bottom two layers make the system an open system: every low–level routine can be replaced with another, provided it remains possible to create the conversion routines.

The upper layer of the K3 User Program provides the suppliers with an environment in which their applications can run. The supplier's application consists of a configured sector overlay. Sector overlays can be used by various suppliers in the same sector, but of course it is also possible for the supplier to write his own sector overlay. Sector overlays are explained in more detail in section 4.5.

Because we want the sector overlay to be operating system and platform independent, we can't use the overlay functionality offered by a specific platform. What is needed is a format for the sector overlay that can be used on every platform. Since we also want to be able to store routines (i.e. program fragments) in the sector overlay (see section 4.5), we decided to use an interpreter for these routines. After looking at several interpreters, we decided to use the bwBASIC 2.10 interpreter, which is freely available on the Internet. Details about our choice for this interpreter and the way it works can be found in chapter 6.

Most conversion routines, but not all, are available to the sector overlay programmer. We decided to use the prefix 'K3' for these routines. From now on, when we speak of a K3–call, we mean a conversion routine available to the sector overlay programmer. Of course, these routines could also be used by the K3 User Program itself, when needed. The K3 User Program provides sector overlay programmers with an interface to these routines by means of a BASIC function. As an example, Figure 4.2 shows how a K3PlayWAV command (see section 4.4.1) in a sector overlay command is executed.

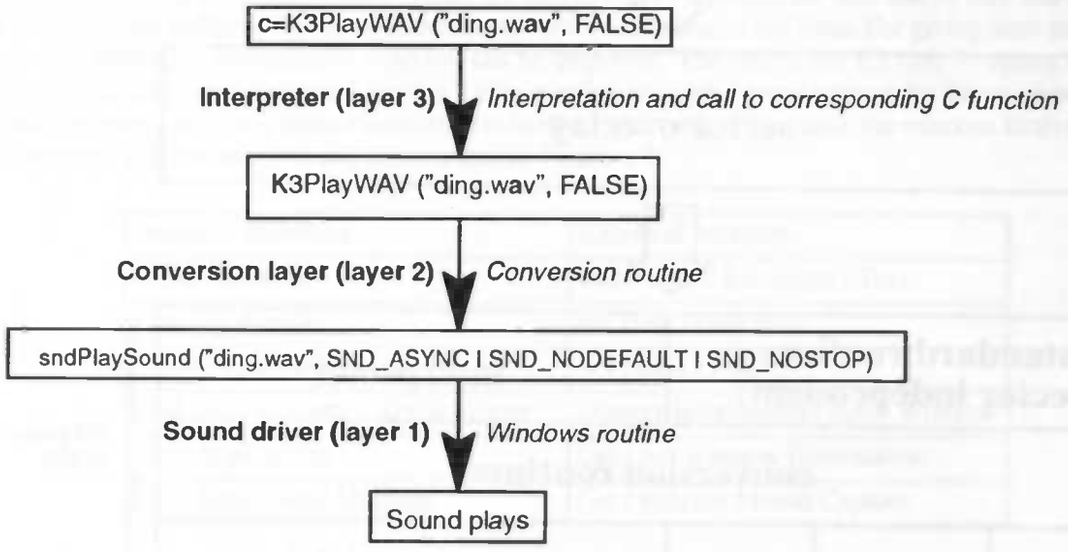



**Figure 4.2:** *Execution of a K3PlayWAV command in a sector overlay routine*

We've already established the need for a network. It doesn't really matter what protocol is used: a conversion routine can be written for almost any protocol. In the demonstration program, no network support is included, but the program is constructed in such a way that it should be fairly easy to add it.

Besides being a way to facilitate easy information updates for the suppliers, the network will also be used to add the client's characteristics to the supplier's database. With the information from all the clients, the supplier constructs the IPI. This process can be highly automated, as is described in section 8.3 and section 8.4. The IPI is a part of the standard routines in the third layer.

### 4.3.4 Implementing the User Program

For the implementation of the user program, we decided to use the library facility provided by Borland C++. In the individual libraries, routines are created to access the data–types. In this way, data–types can be changed internally without any need for change to the calling routine. This is according to the way Parnas defined *Information Hiding* and *Abstract Data Types* [1]: certain properties of a component (in this case: a library) should not be visible in the interface, so that these properties can be modified later in an easy way. The interface, in this context, is called an Abstract Data Type.

In the demonstration program, we used the library facility as a tool to implement abstract data types. They also made it possible to work together on different parts of the program, with new functionality, created by one of us, being adopted by the others as it became available. This approach proved to work very well, as was already known from the previous occasions we worked together.

The K3 User Program, like any other computer program, makes changes to data structures. For platform independence, it is needed that every data structure that is used in the third layer is stable. With this, we mean that the data types have to be standard C data types or special 'K3' data types. The latter types have to be mapped to platform–dependent types in the conversion layer. For instance, we can't use the types provided by RenderWare, since we may not have them available on a new target platform. To overcome this problem, the K3 data types were introduced.

In some cases, the K3 types could be *#defined* to be types provided by Windows. The type **K3Window**, for instance, is 'just' a **HWND** (see [2]). In other cases, the K3 data type is a bit more complicated than the types provided. The **K3Object** data type is an example of this. It consists of a **RwClump** (this is a RenderWare data type, see [3]) and several other fields.

## 4.4 K3 User Program Components

In this section, we describe in some detail the various components of the K3 User Program. Some of these components are so large, that entire chapters are devoted to them to explain them in detail. For these components, we only give a summary of the functionality in this chapter. The other components are explained in full detail.

First, we'll summarize the components:

- **bottom layer and conversion routines:**
  - windowing system
  - 3D rendering engine
  - sound driver
- **standard routines:**
  - Intelligent Personality Identification (IPI)
  - Intelligent Behavior Identification (IBI)
  - Sales engine
- **administrations:**
  - overlay management
  - scenario administration
  - variable administration
- **route planning and motion of 3D objects**
- **interpreter**

Of these components, the interpreter has already been described in this chapter. For a more thorough description we refer to chapter 6. The other components will be described in the remainder of section 4.4.

### 4.4.1 Layers 1 and 2: the 'low–level' drivers and conversion routines

For every component, we will give a description of the desired functionality on the third layer of the K3 User Program. Where necessary, implementation details are given. As an example, the implementation of the sound driver is completely discussed, including the functionality provided by the platform on which the K3 User Program demo was built.

**Windowing system**

The windowing system has to provide the K3 User Program with both internal and external routines. The internal routines can only be used by the User Program itself, while the external routines are also available for sector layer programmers by means of BASIC functions. By standard, the external routines have the prefix 'K3'. The routines will not be described in full detail here. For more information about the implementation, see [4]. In this report, we will only give a high–level description of the functionality that should be provided to the upper layer of the K3 User Program, starting with the internal routines.

First of all there should be window handling routines for creating, destroying, showing and updating windows. Then, the K3 User Program also needs routines for timer management. They provide the upper layer with the possibility to start several numbered timers, each with their own interval. Since the K3 User Program runs in a windowing environment, it may also be necessary to have a call to stop it through a windowing system call. Lastly, the K3 User Program needs routines for sending and getting event messages.

The external routines handled by the windowing system include routines for drawing lines, rectangles and text in the K3 User Program window. There are also routines for setting the font and its size and routines for setting brush and pen. The latter determine the color of rectangles and lines. For giving error messages and other messages, information windows can be displayed. The size of the K3 User Program window can be determined and changed. Lastly, it's also possible to toggle mouse capture for the window. When mouse capture is on, every mouse movement is handled, even when it's outside the window. Both external and internal window routines are summarized in Figure 4.3.

| **Internal Routines** | **External routines** |
|---|---|
| Create Window | Draw Line / Rectangle / Text |
| Destroy Window | Set Font / Font Size |
| Show / Update Window | Set Brush / Pen |
| Start / Stop / Pause / Restart Timer | Display Information / Error Window |
| Stop Application | Get / Set Window Dimensions |
| Get / Send Event Message | Get / Release Mouse Capture |

**Figure 4.3**: *Internal and External window routines in the upper layer of the K3 User Program*

### 3D Rendering Engine

In the K3 User Program, the 3D world is called the scene. In this scene, objects and lights are present. Objects are added to the scene through the object administration. Some of the objects in the scene can act as the *ego object*. There can only be one ego object at any given time. This object can be moved around the scene by the user. The lights can be set, turned off, dimmed and removed by means of sector overlay commands.

In the scene, there are three cameras at any given time. One of these cameras is set at a fixed position in the scene. This position can be changed by the sector overlay. Two cameras are attached to the ego object: every time the ego object is moved, these cameras are moved as well. One of these cameras is showing what the ego object is seeing (eye–view). The other camera is following the ego object at a fixed distance.

All information about the object is stored in the object administration. This not only contains physical information about the object, but also information about its position and all the extra information that is contained in an *extended RWX file*.

The K3 User Program uses an extended version of the RenderWare RWX format for 3D objects (see [3]). In an RWX file, '#' denotes a comment. To the K3 User Program, lines starting with '#@' are no comments. Rather, they are treated as lines with extra information about the object, like the position and direction of the eye–view camera. More information about the extended RWX file format can be found in chapter 5.

Next, we will give a short description of the 3D rendering routines that can be found in the top layer of the K3 User Program. The routines are summarized in Figure 4.4.

*Scene routines.* These are routines for creating and destroying scenes and for rendering the scene to the window. There's also a routine for controlling automatic rendering. Automatic rendering is done by the K3 User Program every 20 milliseconds (using a timer). All these routines are external.

*Camera routines.* There are routines for moving the camera, setting it at a given position and changing its direction (pan, tilt, revolve) and also for setting and getting the direction (look–at, look–up, look–right). These routines all work on the fixed camera. As we have mentioned earlier, this is the only camera that can be moved around by the sector overlay. The other two cameras are attached to the ego object. For all the three cameras simultaneously, a background bitmap can be set and there are routines for setting and getting the ego object as well as routines for getting and setting the current camera. These routines are all external. Lastly, there are internal routines for creating and destroying the cameras.

*Light routines.* These are the routines for setting, moving and destroying lights and setting them at a given brightness. There are three types of lights: point lights (which illuminate in all directions), directional lights and conical lights. All these routines are external.

*Object management routines.* Routines for reading objects, adding them to the scene and destroying them. The objects can be scaled, duplicated, moved, set at a given position and rotated. It's also possible to rotate part of the object (identified by means of a tag). Furthermore, objects can be set on the floor and have their height changed. The K3 User Program also provides for changing the color, reflections and opacity of the objects and can detect collisions between objects. Lastly, it is also possible to calculate the distance from any object to a certain point in the scene. All the object management routines are external.

*Calculation routines.* Since the 3D routines use the type **K3Real** (see [4]), we can't use the normal C routines for calculations: we need routines for performing calculations on them. The K3 User Program provides routine for addition, subtraction, division and multiplication. Besides these, there are also routines to convert from and to standard C types **int** and **float** and there is a routine to convert a triple of floats to a **K3Coord**, which denotes a coordinate in the scene. All calculation routines are external.

*Miscellaneous routines.* There are routines for initializing and destroying the 3D library. These routines are internal.

| | |
|---|---|
| **Scene Routines** | Create / Destroy Scene<br>Render Scene<br>Start / Stop Automatic Rendering |
| **Camera Routines** | Move to / Set at Position<br>Pan / Tilt / Revolve Camera<br>Get Look–At / Look–Up / Look–Right<br>Set Look–At / Look–Up / Look–Right<br>Set Background<br>Get / Set Ego Object<br>Get / Set Current Camera<br>Create / Destroy Cameras (internal) |
| **Light Routines** | Set Point / Directional / Conical Light<br>Destroy Light<br>Set Light Brightness<br>Move Light |
| **Object Management Routines** | Read Object / Add to Scene / Destroy<br>Scale Object<br>Duplicate Object<br>Move Object / Set At Position<br>Rotate Object / Part of Object<br>Set Object height<br>Set Object On Floor<br>Set Object Color / Reflections / Opacity<br>Collision Detection<br>Distance of Object to Point |
| **Calculation Routines** | Add / Sub / Div / Mul<br>Int2Real / Real2Int<br>Float2Real / Real2Float<br>Floats2Coord |
| **Miscellaneous Routines** | Initialize / Destroy 3D Library |

**Figure 4.4:** *3D Rendering routines in the top layer of the K3 User Program. All routines are external, unless specified otherwise*

## Sound Driver

As an example of what the implementation of the K3 User Program looks like, we describe the sound driver in full detail. The other program components are also described in detail in [4]. In this document, more information can also be found about the files in which the sound driver routines are stored.

The K3 User Program adopts the Microsoft WAV format for digitized sounds. The following routines are to be made available in the third layer by layer 2:

*int K3PlayWAV (char *, int):* a routine to play a WAV file. The first parameter is the file name, and the second parameter indicates whether the call should return immediately after the sound is started or if it should wait until the sound is finished. In the latter case, the parameter will be set to FALSE. Otherwise, it will be TRUE. The return value is TRUE when the sound could be played, FALSE otherwise.

*int K3LoopWAV (char *):* a routine to repeatedly play the WAV file with the specified file name. The routine returns immediately after the sound is started. The return value is TRUE when the sound was indeed started, FALSE otherwise. The sound should be stopped using the *K3StopWAV* call.

*int K3StopWAV (void):* a routine to stop the currently playing sound. Returns FALSE if there was no sound playing when the function was called, TRUE otherwise.

These three routines can be implemented using only one Windows API call: the **sndPlaySound** call (see [2]). The following functions, which implement the three routines described above, are added to the conversion layer:

```
int K3PlayWAV (char *filename, int wait)

{  if (!wait)
     return (sndPlaySound (filename, SND_ASYNC | SND_NODEFAULT | SND_NOSTOP));
   else
     return (sndPlaySound (filename, SND_SYNC | SND_NODEFAULT | SND_NOSTOP));
}

int K3LoopWAV (char *filename)

{  return (sndPlaySound (filename, SND_ASYNC | SND_NODEFAULT | SND_LOOP | SND_NOSTOP));
}

int K3StopWAV (void)

{  return (sndPlaySound (NULL, 0));
}
```

The sound driver is already contained in Windows 95, so there is no real sound driver library added to the bottom layer of the K3 User Program for sound support. Still, we thought it would be a good idea to put the sound support in a separate library, since this simplifies program maintenance. Besides that, it also makes it easier to port the program to another platform. For a description of how these routines are activated from within a sector overlay program, see Figure 4.2.

### 4.4.2 Standard routines

Now, we will describe the standard routines in the top layer of the K3 User Program. These routines form the heart of the program, because together they perform all actions normally done by a salesman. A salesman analyzes his customer and adjusts his sales tactics to the result of his analysis. And then, while he is executing his tactics, he carefully watches his customer to see if his tactics needs more adjustments. This whole process is shown in Figure 4.5. We have tried to make the K3 User Program behave in the same way, as will be explained below.

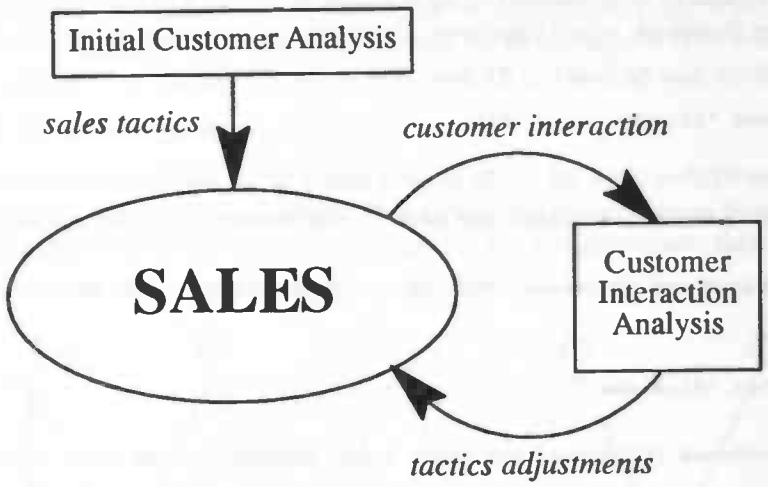



**Figure 4.5:** *The sales tactics are adjusted according to the customer's characteristics. This process continues even after the initial analysis by means of interaction analysis*

### Intelligent Person Identification (IPI)

The task of the IPI is to do the Initial Customer Analysis shown in Figure 4.5. To be able to do this, a salesman needs to know certain information about the customer, which he will either look up in the company client database or ask for himself (and after that he may add the information to the company client database). The K3 User Program can do the same, when it is provided with a questionnaire from the supplier.

Besides the possibility to ask for certain information, it could also be feasible to obtain this information from other databases. This process is called data mining. Of course, this can only be done if the sector overlay knows where to get the data. A kitchen supplier, for instance, could query the database of a building contractor or a house–agent for the measurements of the kitchen and the positions of the power–points. When this information can not be obtained in this way, then the customer can be asked to provide the information.

The IPI is implemented as a series of neural networks, with the answers to the questionnaire as inputs. How this is done, can be seen in sections 8.3 and 8.4. The result of running the answers through the networks is a set of client characteristics, which is stored by means of variables in the K3 User Program's variable table. These variables are further processed by the Sales Engine, as will be explained later.

The K3 User Program's variable table is implemented by means of a hash table, with the identifiers as hash keys. Hashing is a technique which can be used when the number of keys actually stored is small relative to the total number of possible keys. In this case, hash tables become an effective alternative to directly addressing an array, since a hash table typically uses an array of size proportional to the number of keys actually stored. Instead of using the key as an array index directly, the array index is computed from the key by means of a *hash function*. Cormen, Leiserson and Rivest ([6], chapter 12) provide an excellent introduction to hashing and hash functions.

The K3 User Program's variable table is a global hash table, which can be used by other components when needed. The variable table is a separate library, with functions to store and read numerical and alpha–numerical data. The IPI uses numerical variables to store the client characteristics.

### Intelligent Behavior Identification (IBI)

The purpose of the IBI is to perform the Customer Interaction Analysis. A salesman will carefully look at the customer and his actions to find out if the strategy he has chosen is still right. As an example consider

a salesman who erroneously estimated that his customer would like to buy a relatively cheap kitchen. When the customer wants a dish–washer and a microwave oven in his new kitchen, the salesman sees his mistake and he will show the customer a more expensive kitchen. In this case, it was the customer's wish for expensive kitchen equipment that caused the salesman to change his strategy.

In the K3 User Program, the user probably selects the kitchen equipment himself. This action needs to be identified by the IBI as being a possible cause for changing the sales strategy. We did not implement this feature in our demonstration program, but we will describe how this could be done in section 8.5. For now, it is enough to know that the conclusions the IBI comes up with, are stored in variables just like the IPI results.

### Sales Engine

The Sales Engine is the part that links it all together: it will call the IPI to establish an initial sales strategy, then it will execute this strategy while it is evaluating user actions in a loop. Here, evaluating user actions means calling the IBI.

The Sales Engine determines the sales strategy by means of a small rule base. This rule base uses the variables set by IPI and IBI to determine a strategy. A strategy can consist of several *scenarios*, which are placed on the *scenario stack*. The Sales Engine 'plays' these scenarios, which are written in the interpreter language. In the case of our demonstration program, this is the BASIC language. Figure 4.6 depicts the sales process as it is implemented in the K3 User Program.

The sales engine is implemented as a semi–fuzzy inference engine with rule base. The inference engine evaluates the rules in the rule base. Rules in the rule base have the following form:

IF var1 = fuzzy_value1 [AND var2 = fuzzy_value2 AND ....] THEN some_scenario

There are five possible fuzzy values: Negative Big (NB), Negative Small (NS), Zero (ZO), Positive Small (PS) and Positive Big (PB). For all rules, an activation level will be calculated. All conclusions (scenarios to be played) with an activation level above a certain threshold (typically about 0.8) will be put on the scenario stack, the scenario with the highest activation level first. The activation level of a rule is the lowest *membership value* for the conjuncts in the condition part of the rule.

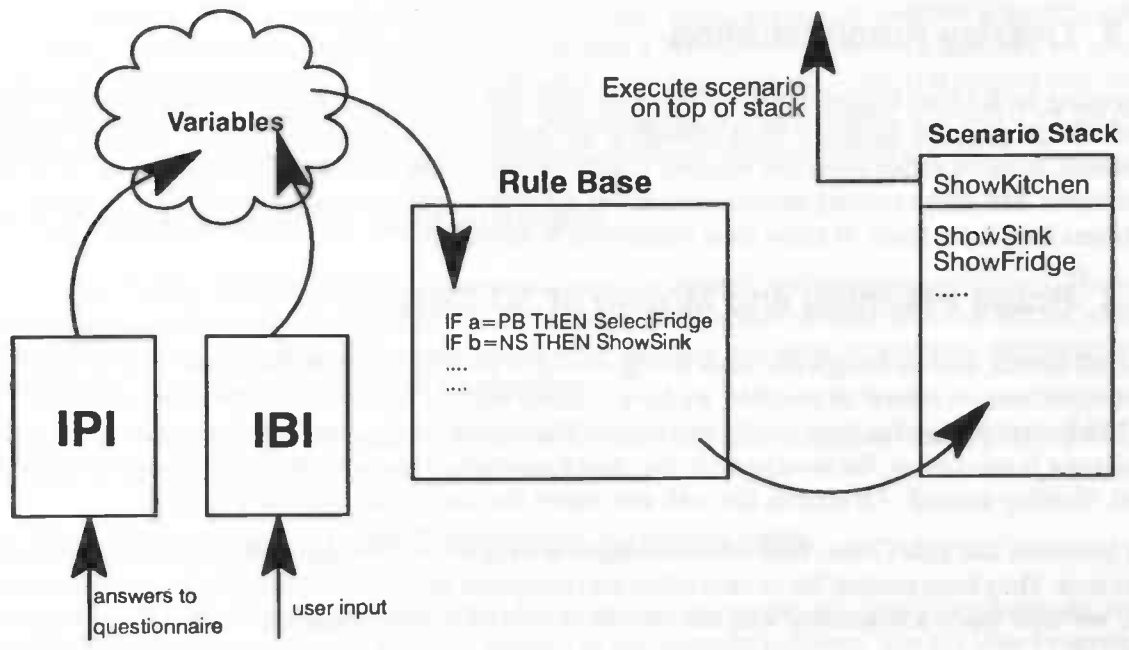



**Figure 4.6:** *Implementation of the sales process in the K3 User Program*

Membership values have to be calculated for every conjunct in the condition part of the rule. In Figure 4.7, the membership functions for these calculations are shown. As an example of how an activation level for a rule is computed, consider the variables a and b, with a=0.75 and b=0.65 and the following rule:

IF a=PS AND b=ZO THEN show_sink

The membership function for Positive Small yields a value of 0.5 for input a=0.75 and the membership function for Zero results a value of 0 for b=0.65. This means that the activation level for this rule is 0, since this is the minimum value of 0.5 and 0. Consequently, the scenario **show_sink** will not be placed on the scenario stack.

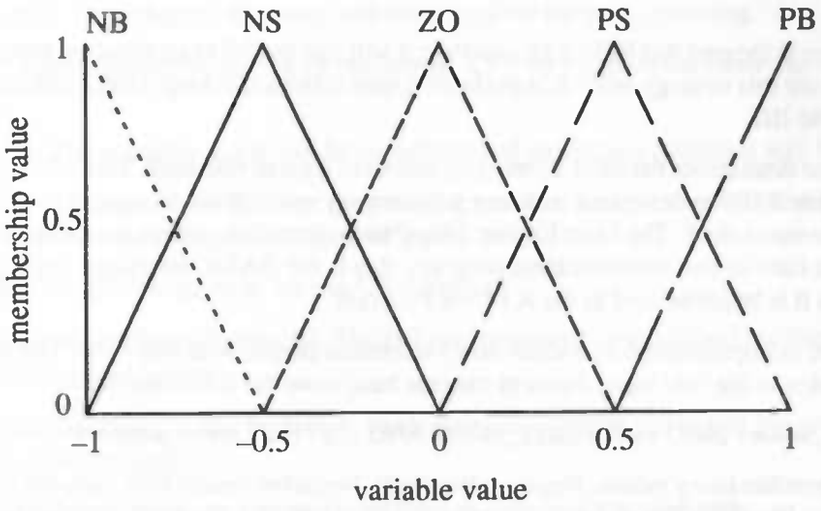



**Figure 4.7**: *Membership functions for the Semi–fuzzy inference engine*

### 4.4.3 Overlay Administration

When using the K3 User Program, it will frequently occur that one supplier is visited often for a reasonable period. To prevent the program from reloading the entire sector overlay on every visit, a cache is introduced. Besides sector overlays, supplier configurations could also be cached, as well as 3D object descriptions. The entire overlay administration was not implemented in our demonstration program, but provisions have been made to allow easy integration in the future.

### 4.4.4 Route Planning and Motion of 3D Objects

In the 3D scenes, human beings and other living creatures are often present. When we want to make the whole experience as natural as possible, we have to make sure the human beings in the scene do not only look like human beings, but they should also behave like human beings. An important aspect of this natural behavior is movement. We have to make sure that the people in the scene walk like real people, instead of just 'floating around'. Of course, this will also apply for other living creatures like pets.

Hans Seinhorst and Eric Grave, both former students at the University of Groningen, have done research in this area. They have created the motion editor and composer *MedCom* [5]. With the help of Hans Seinhorst, we have made a beginning with the natural motion of human objects in the K3 User Program's scene.

Whenever the user clicks with the right mouse button in the scene, the ego object will walk to the position that was clicked. For this, first of all a route has to be determined to the specified position. After this,

Hans's routines are called to make the object move to this position. Eventually, the object will walk like a real human being. However, at the moment of writing this has not yet been implemented. A detailed description of route planning and motion of 3D objects can be found in chapter 7.

## 4.5 Sector Overlays

In the remainder of this chapter, we will give an overview of the contents of a sector overlay and how such an overlay could be configured by a specific supplier. As we have established before, a sector overlay can be used by several suppliers in the same line of business, so there has to be a possibility for a supplier to configure the overlay. This configurations, then, makes the sector overlay behave in the way the supplier sees fit. In section 4.5.2, we will show how a sector overlay can be configured, but now we will continue with a description of the sector overlay contents.

### 4.5.1 Sector overlay contents

In the previous part of this chapter, we have already seen most of the things that go in the sector overlay. We will summarize the entire sector overlay contents here, with a short description of each item.

*Scenario Names.* For administration purposes, a list with all valid scenario names is present. This is used, amongst others, to check the consistency of the sector overlay.

*Sales Engine Rule Base.* The list of rules, used to determine the scenario that has to be played.

*IPI neural network.* The description of the neural network that implements the IPI.

*IBI.* The list of rules, used to implements the IBI.

*Questionnaire for the IPI.* The questionnaire, whose answers are used to feed the IPI.

*Overlay initialization.* A BASIC routine for overlay–specific initialization.

*Scenario routines.* These routines implement the scenarios. For every scenario, a BASIC subroutine exists with the same name. This subroutine is called when the scenario has to be executed.

*Action handlers.* Action handlers are BASIC subroutines which are used to perform a specific task on a 3D object. In the extended RWX–file (which contains a 3D object description), lines can be present to tell the K3 User Program which routine to call when that object (or a sub–object) is clicked with the right mouse button. The routine that is called is the action handler. As an example, consider a sector overlay for kitchen sales. In this overlay, a routine could be present to open and close cupboards.

### 4.5.2 Sector overlay configuration

The sector overlay can be configured by a specific supplier through configuration parameters. The overlay initialization routine will ask the supplier for these parameters, which are then stored as BASIC variables. The scenario routines and the sales engine rule base use these variables. In this way, the sector overlay can behave quite differently for different suppliers.

Besides the configuration parameters, the sector overlay will also download the supplier's product descriptions and other supplier–specific items like logos.

## 4.6 Internal K3 User Program processes

When a client contacts a supplier, the latter will send a sector overlay ID. This ID contains the name of the requested sector overlay and a version number of the currently used one. The K3 User Program starts a search in the local sector overlay database. If the client has contacted the supplier before, the sector overlay will be stored in this database. Otherwise, the K3 User Program asks the supplier to send the sector

overlay. This sector overlay will be stored in the sector overlay database. Next, the K3 User Program asks the supplier to send the sector overlay configurations. At this moment the K3 User Program can be initialized. There are two kinds of initialization, namely the system and sector initialization. The system initialization initializes the standard components. These components are:

- Window environment: The K3 User Program window will be defined and displayed on the screen.
- 3D: The 3D–routines will be initialized and coupled to the K3 User Program window.
- Route planning: Variables are allocated and initialized for the route planning.
- Interpreter: Variables are allocated and initialized for the route planning.
- Timer: The timer will be started. In the current K3 User Program the timer causes an event every 20 ms.
- Scenario: The scenario–stack will be initialized and the default scenarios will be pushed onto it.

The sector initialization gives the K3 User Program the look and feel the supplier wishes. The following elements are initialized using the following configurations:

- RuleBase: loads the rulebase for the Sales Engine
- IBI: loads the rulebase for the IBI. The IBI pre–processor, as described in chapter 8, is incorporated in the event handler.
- IPI: reads the client profile from disk, reads the questionnaire and builds the neural networks retrieved from the supplier.
- Overlay: The supplier can set some variables to influence the sales talk.

Next, the Init–routine of the sector overlay will be started. In this Init–routine the supplier can define, for example, his own 3D environment. After the execution of the Init–routine the IPI will be called. The IPI analyzes the client, deduces a sales strategy and stores this sales strategy in the global variable administration. Then the Sales Engine is called and the sales talk is started. This Sales Engine fires its rulebase and the first scenarios are added to the scenario stack. During the Init–routine and the execution of the scenarios, products and environments are inserted in the 3D virtual world. If the client has contacted the supplier before, products, scenarios and environments are stored in local databases, like the sector overlay. The product descriptions, scenarios and environments contain a date, at which they have been created. If this date is still valid the data in the databases can be used. Otherwise the data has to be retrieved from the supplier. Further action of the program depends on the event handler. Figure 4.8 depicts the K3 User Program processes.

An important event is the timer event. To be able to handle more tasks at once, like moving several objects and rendering at the same time, we introduced a timer. A timer causes a timer event every few (pre–defined) milli–seconds. By using a timer, the K3 User Program is able to do multiple tasks simultaneously, thus creating a multi–tasking environment. At each timer event, several components of the K3 User Program execute one step of their process. The following processes are running using the timer:

- rendering the 3D virtual world.
- playing the sales talk: executing the routine that is at the top of the scenario stack.
- executing one motion of all moving objects.
- executing interpreter steps (top of execution stack).
- sales engine rulebase
- IPI

When executing several processes using a timer, synchronization problems may occur. The timer, for example, could give a new signal before the previous step of a particular process is ready. To solve these problems we introduced semi–semaphores.

Another set of important events are the events caused by the user. In the current K3 User Program, all the user input is handled by Windows 95. Windows 95 translates the hard user input into WM_ events. If the K3 User Program has to run under another operating system, some routines have to be written to translate the hard input to WM_ like events.

The reactions to the received input are currently hard–coded. In the future it has to be possible that each supplier can define his own routines for every user input. However, it's wise to make the control of the ego object standard. In this way all the applications, running on the K3 User Program, will be consistent.

All the non–trivial input is handled by routines. In this routines the input will be translated into actions. For example, a right mouse click at an object x will be translated into "object x has been clicked". The action to be performed will be executed. In this particular case, the ego object has to walk to the clicked object and if the supplier has developed an action routine for this object, this routine has to be executed (i.e. the routine is put on the execution stack). Next the IBI will be called with the message "object x has been clicked". When the IBI writes some new results into the variable administration, the flag SALES_MAY_CHANGE will be set. If the changed or new variables are part of the variables used by the IPI, the flag IPI_MAY_CHANGE will also be set. These flags will be checked at each timer event.

The IPI will be called if the IPI_MAY_CHANGE flag is set and the Sales Engine is called when the SALES_MAY_CHANGE flag is set. After calling these components the flags are reset. So if something is changed in the variable administration, this can cause a new sales strategy and changes in the sales talk (like starting new scenarios). We use flags to give other processes, or even new processes, the opportunity to influence the sales talk by just setting these two flags. If the IPI determines a new sales strategy it will be changed in the variable administration. Next the IPI has to set the SALES_MAY_CHANGE flag to activate the Sales Engine. An activation of the Sales Engine may cause changes in the sales talk. At implementation level this means that scenarios are added to, deleted from or moved within the scenario stack (see also chapter 5). The Sales Engine also decides when the client is ready for ordering products. An ordering–scenario will be added to the stack and will be executed when it reaches the top of the stack. Like all other scenarios, a supplier can develop his own ordering–scenario.

When no scenario is running, the user controls the ego object. When doing so, he can activate the IBI by doing interesting actions in the scene. As said before this can cause changes in the sales talk and probably new scenarios will be fired etc.. The user can also terminate the K3 User Program.

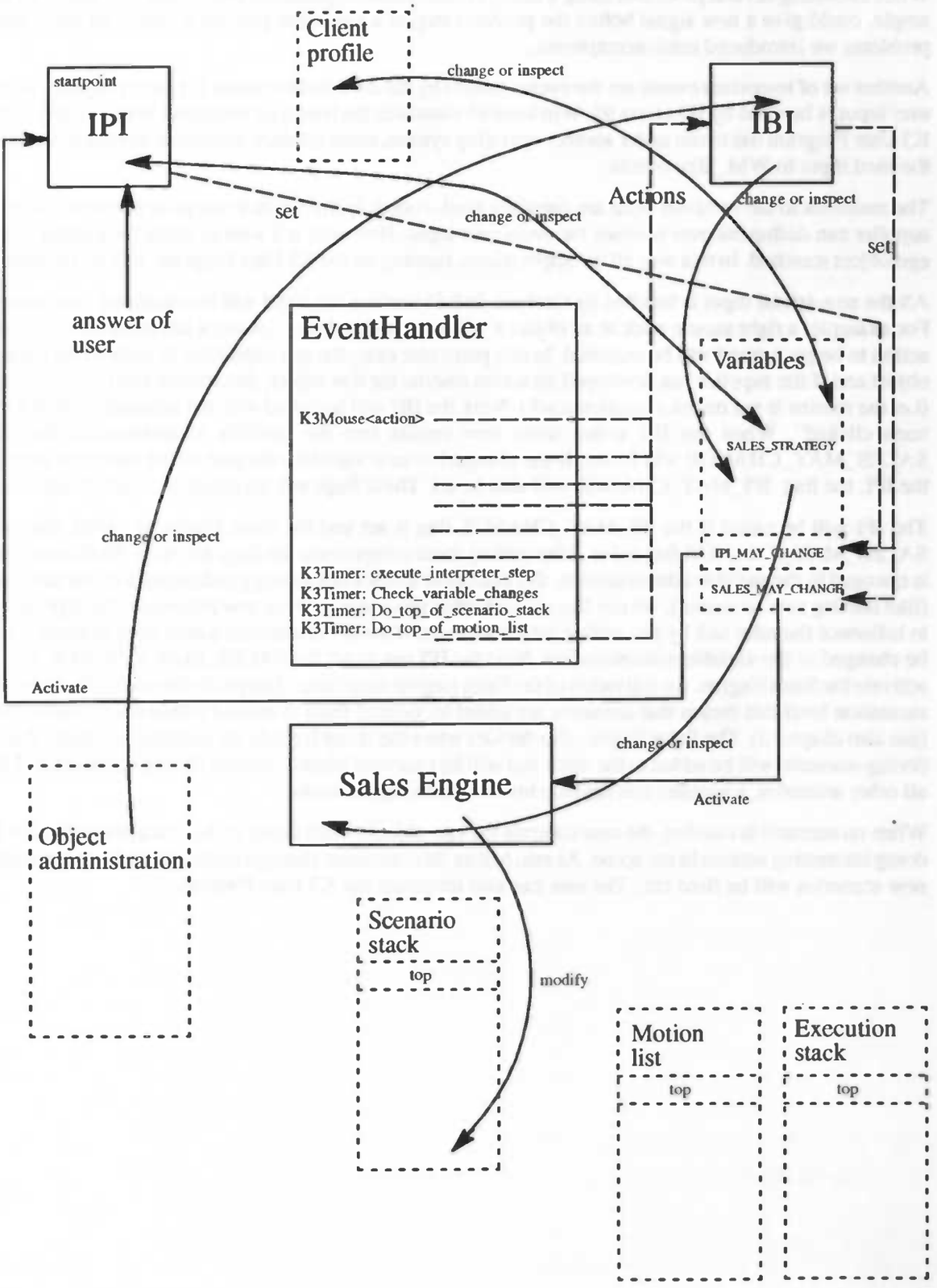



**Figure 4.8:** *The K3 User Program process.*

## 4.7 Ego objects

During the sector overlay initialization, the supplier has to set one of the objects in the virtual world to be the ego object. The ego object represents the client in the virtual world. When using the mouse, the client can move the ego object around. The way the client observes the virtual world will depend on the choice of the viewpoint. The client is offered three different viewpoints. He can observe the world using the eyes of the ego object. This will look like walking in the world by yourself. Secondly, he can observe the world by flying just behind the ego object. And thirdly, the supplier can define a viewpoint somewhere in the virtual world. By using this static viewpoint, the client can see the ego object moving in the world. During the sales talk the client can change from viewpoint.

The ego object can be in the free walk–mode or the sales–mode. If the client can control the movements of the ego object, the ego object is in the free walk–mode. Otherwise, the K3 User Program controls the ego object. In this sales–mode the client can still change the viewpoint. For example, the client gets a guided tour in a kitchen. The supplier can show the client all nice and handy applications in the kitchen he wants to sell. The ego object will move from application to application.

In the K3 User Program, there's only one ego object at a time. However, during the sales talk all other objects can become ego object. In the current K3 User Program, only human beings can move in a natural manner. All other kind of objects will 'slide' through the virtual world.

## Chapter References

[1] D. Parnas, "On the criteria to be used in decomposing systems into modules", *Communications of the ACM, vol. 15, no. 12, 1972,, pp. 1053–1058.*

[2] Microsoft, "Windows 3.1 API reference"

[3] Criterion Software Ltd., "RenderWare Programmers Reference Manual"

[4] R. Buist, J. de Graaf and W. Wichers, "K3 User Program Implementators Guide", *University of Groningen, August 1996.*

[5] H.A. Grave and J.H.M. Seinhorst, "Fuzzy–Based Motion Editing and Composing", *Master's Thesis, University of Groningen, August 1995.*

[6] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, "Introduction to Algorithms", *The MIT electrical engineering and computer science series, The MIT Press.*

# Chapter 5 Administrations

This chapter describes the implementation of the data structures and functions for storing and manipulating the data in the K3 User Program. For the implementation of the data structures we have chosen for abstract data types. The first section explains what abstract data types are and why they are used. The following sections describe the administration for the objects, the variables, the scenarios and the overlay management.

The first section describes the administration for the three–dimensional objects used in the scene. The second section describes the administration for the scenarios. The third section describes the administration of the variables used throughout the program. These variables are set by the client profile and the IPI, and can be used by the fuzzy rule bases to determine a sales strategy, the next scenarios or to adjust the objects to the liking of the client. The fourth section describes the administration for the management of the overlays. An overlay contains the information the provider has to send to the client for the program to work, like scenarios, rules for the rule bases or new functions to augment the current set of functions.

## 5.1 Abstract Data Type

An abstract data type (see [1]) is a mathematical object. It is defined by means of its properties. The difference between an abstract data type and other data types is that the implementation of an abstract data type is not directly available. In case of an abstract data type there are only a number of operations available on the conceptual variables of the abstract data type. So an abstract data type abstracts from the implementation of the data structure and provides a set of operation to manipulate the data.

The advantage of working with abstract data types is that the implementation of the data structure can change without having to make changes to the rest of the code. The only changes that have to be made, are to the data structure and to the functions that access that structure, but the function calls and therefore the interface stay the same. The data structures described in the following sections can only be inspected and altered using the appropriate functions.

## 5.2 Object administration

The objects we are talking about in this section are the 3D objects used in the scene. It is not sufficient to only have the 3D representation, we must have more information about the objects in order to perform operation on these objects. We might want to change the position of an object in the scene, display some information about an object or know if we are looking at the front or back of an object. For this purpose we have created the object administration.

Before we give an overview of all the things we administrate, we will describe how objects are constructed. Objects are described in the RenderWare RWX format in terms of lines, polygons, surfaces

etc. But an object can be built up with several sub–objects. For example, when we have a cupboard object like in Figure 5.1, the drawers, the door and the main body of the cupboard are called sub–objects. When we tag the sub–objects, we can address each sub–object individually and perform operations on these sub–objects. When we move the main body of the cupboard, we want the drawers and the door to move also. Therefore, the sub–objects are organized into a hierarchy where an operation on a sub–object is also performed on all its descendents in the hierarchy.

Opening a door is a completely different routine from opening a drawer. So each tagged object has to have its own routines. In Figure 5.1 we have a cupboard that consists of four sub–objects: a main body, two drawers and a door. All these sub–objects are tagged with a unique number. The main body of the cupboard has three descendents in the hierarchy: the two drawers and the door. If we were to move the main body of the cupboard, we would also move the door and the two drawers.

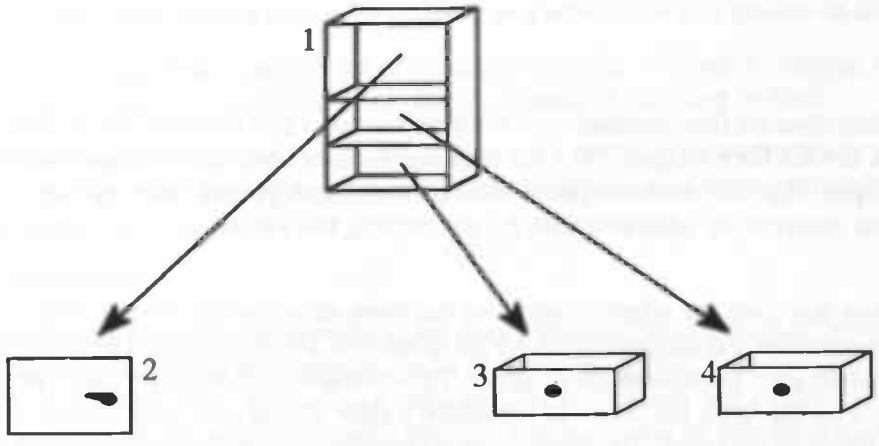



**Figure 5.1**: *The cupboard object consists of four sub–objects: a main body, a door and two drawers. All objects are tagged with a unique number.*

We could also imagine that we want to be able to perform two actions on a tagged object. For example, clicking on a drawer with the left mouse button opens a drawer, while double clicking on a drawer with the left mouse button removes a drawer. Therefore we have to administrate which actions to perform on which input. This has to be done for each tagged object. Furthermore, if we click on an closed drawer with the left mouse button, we want it to open. But if we click on an open drawer with the left mouse button, we want it to close. Therefore, we must record the state each tagged object is in.

## object administration items

We will now give a description of all the information we administrate for each 3D object. Not all information stored here is useful for each object. A *product group* is not useful for a humanoid object, but it is much easier to use a standard administration for all objects than to use a different form for each type of object. Some of the information can be derived from other information, for instance the *bounding box* can be calculated each time it is needed. But calculating the bounding box is a recursive procedure that takes a lot of time. Therefore it is much faster to administrate this information and adjust it each time the size or position of an object changes than to calculate it each time.

*3D object*. This gives us access to the 3D object. The specific type of object depends on the 3D engine.

*Name*. This is the name of the product. This name can be used to give more information about the object and this name is used by other parts of the system to refer to this object.

*Product group*. This is the name of the product group instead of a specific product name. For example, if the 'Bosch 3000 TD' is a refrigerator, the *name* of the product would be 'Bosch 3000 TD', while the *product group* would be 'refrigerator'.

*Actions*. These are the different action for each of the (sub–)objects. Each sub–object, like a drawer in a cupboard, can have its own action when activated. A drawer has to slide forward, but a door has to rotate.

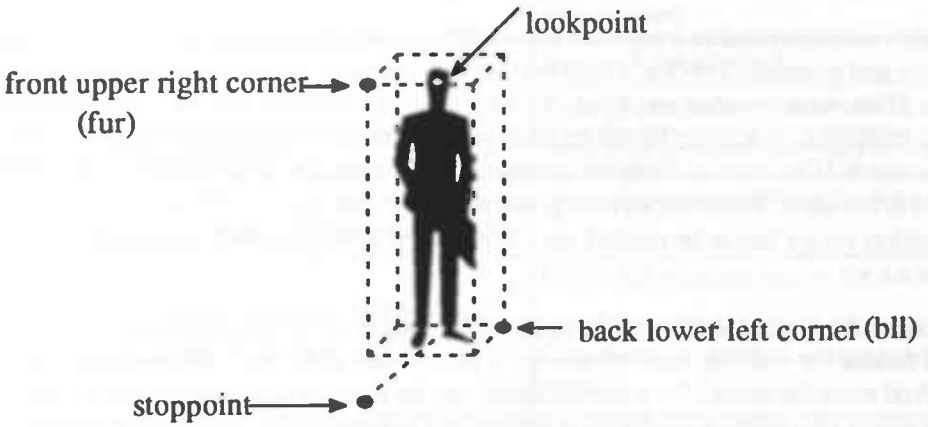



**Figure 5.2**: *The fur and bll define the bounding box for this man. With the given stoppoint and lookpoint, the ego object would walk up to the stoppoint and face the man.*

*Front upper right (fur)* and *back lower left (bll)*. The size of an object is administrated with two coordinates: the back lower left coordinate and the front upper right coordinate. These are actually the coordinates of the smallest box that encloses the object: the bounding box. From these coordinates we can calculate the size of the object and use it for object collision detection in the route planning routines (see chapter 7).

*Lookpoint* and *stoppoint*. If the client can walk through the scene, looking through the eyes of the *ego object*, there are two ways in which he can get to an object of interest. One way is to 'walk' towards the object, using for instance the arrow keys on a keyboard. The second way is to let the computer calculate a route to the object and walk that route. For this automatic way, we need to know where to stand still in front of the object and where to look at. Usually we will stand half a meter or one meter in front of the object, looking at the middle of the object. So we need to have a *stoppoint* (a point in front of the object) and a *lookpoint* (where to look at when standing in front of the object).

*Look_at vector, look_up vector* and *camera position*. In order to use the *ego camera* (the camera associated with the ego object), we have to know the location of the camera, the direction in which to point the camera and where to hold the top of the camera. For this, we have respectively the camera position, the look_at vector and the look_up vector as seen in Figure 5.3. For humanoid objects we put the camera position in the head of the object to get a realistic view. It is as if we were looking through the eyes of the humanoid object.

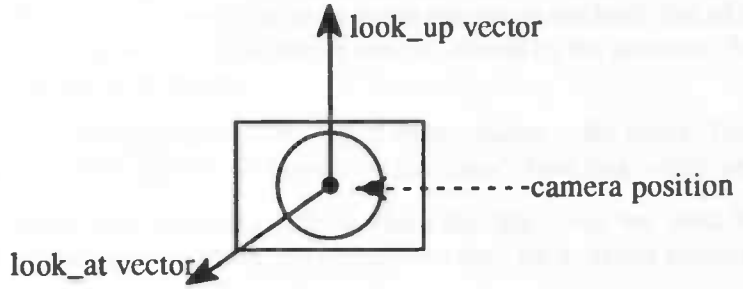



**Figure 5.3**: *The camera position, look_at vector and look_up vectors define exactly how the camera should be placed for the ego object.*

*Magic position* and *magic action*. Each object can have a *magic position*, an invisible area on the ground. When this area is entered by the ego object, the *magic action* is executed. For example, when the ego object is walking through the magic position of a painting, the painting can drop on the ground after which the insurance policy can be altered. Since the magic positions have to be checked each time the ego object

moves, they are also administrated in a separate linked list. The magic action is a bwBASIC routine with an optional rotation and translation vector. These two vectors can have different meanings when used with different routines. If we want to rotate an object, we have to give a rotation axis and a point around which to rotate. Here the rotation axis is given by the rotation vector and the point around which to rotate is given by the translation vector. If we want to translate an object, we have to give a translation vector. In this case, the rotation vector is not used. These vectors are given relative to the origin of the object. When the object is rotated, the rotation vector has to be rotated also. When the object is scaled, the translation vector has to be scaled also.

*Model.* This describes the type of motion we have to use for the object. If we want a dog to move, we have to use a different model for walking than we use for a human being. So for a dog we use a four–legged kind of motion. And even the model for a human being can be divided into a model for a man, a woman and children. We could also make a model for a person in a wheelchair or a person on crutches.

*Motion.* This describes the kind of motion the object has to use, like walking or running. For the ego object, which is usually a person, this could be a walking, running or crawling motion.

*Emotion.* Motion can be coupled to emotion. The motions for people who are happy or people who are afraid are different. This could be used to show how happy people are after buying products from the supplier.

### Object file extension

Since each object has its own object information, we store this object information in the object file. Since we use RenderWare as the 3D rendering engine, we have chosen to extend the RenderWare object description commands. In an RWX file, '#' denotes a comment. We have given '#@' a special meaning: '#@' is followed by a command that provides us with the information needed by the object administration. An overview of all commands is given in Figure 5.4.

| command | description |
|---|---|
| #@ name <*string*> | Name of the product. |
| #@ product_group <*string*> | Type of products to which this product belongs. |
| #@ tagaction <*tag*> <*string*> | Action to perform when tag is activated. |
| #@ tagtranslation <*tag*> (<*float*>, <*float*>, <*float*>) | Gives the translation vector for each sub–object. |
| #@ tagrotation <*tag*> (<*float*>, <*float*>, <*float*>) | Gives the rotation vector for each sub–object. |
| #@ lookpoint <*float*> <*float*> <*float*> | Coordinate to look at when standing in front of object. |
| #@ stoppoint <*float*> <*float*> <*float*> | Coordinate to walk to when automatically walking to the object. |
| #@ look_at <*float*> <*float*> <*float*> | Direction of camera in ego object. |
| #@ look_up <*float*> <*float*> <*float*> | Topside of camera in ego object. |
| #@ campos <*float*> <*float*> <*float*> | Position of camera in ego object. |
| #@ magic (<*float*>, <*float*>)* | Polygon on the floor describing the magic position. The polygon is made up of three or more coordinates. |
| #@ magicaction <*string*> | Action to perform when ego object is in magic position. |
| #@ model | Gives the model of the object like man, woman or wheelchair. |
| #@ motion | Gives the type of motion like walking or running. |
| #@ emotion | Gives the emotion of the object like happy or sad. |

**Figure 5.4:** *Extentions to the RWX file format.*

The administration described above is used when the system is running. Besides this administration, we have to keep track of which objects are present on the hard disk of the user. Because we want to minimize the data traffic on the network, we will store as much objects on the hard disk of the client as the client will let us. But objects may become obsolete or may be altered by the provider. So we have to store the following information for each object:

*Object version number.* The provider sends a list of all his objects to the client. This list also contains the latest version numbers of his objects. All objects on the client's hard disk which are outdated are deleted.

*Date of last use.* This gives us the exact date on which the object was last used. With this date, we can perform some sort of garbage collection. All objects not used for a certain amount of time are deleted.

The check for 'version number' and 'date of last use' will be done during the startup sequence of the K3 User Program.

## 5.3 Scenario administration

Scenarios are the building blocks of a sales talk. These scenarios are written by the provider in the interpreter language and are sent to the client. The interpreter language is bwBASIC augmented with the K3

functions, which will be described in chapter 6. The Sales Engine determines which scenario is played and when it is played.

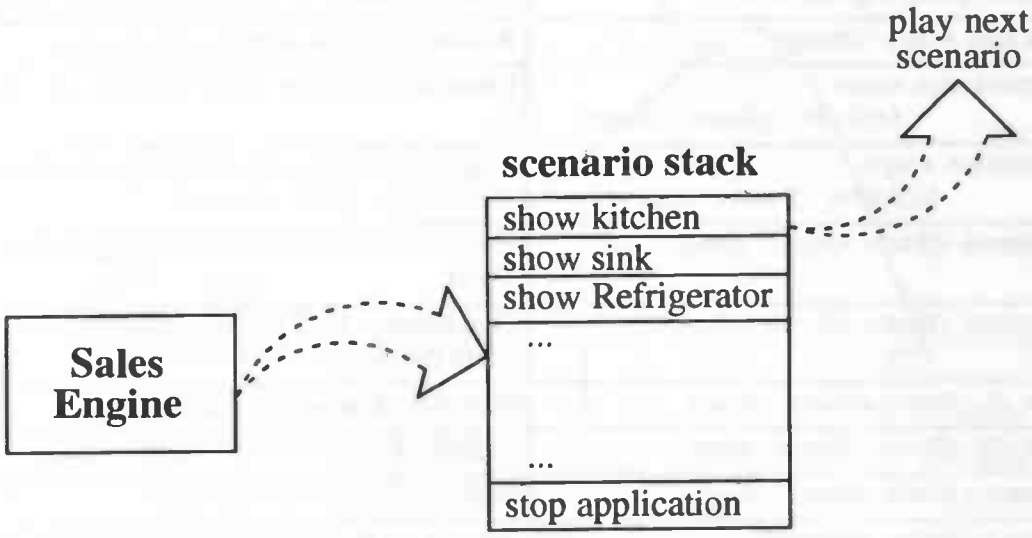



**Figure 5.5**: *The Sales Engine builds the scenario stack, from which the top element is played.*

The Sales Engine places the scenarios on top of the scenario stack. The sales talk is started by playing the top element from the scenario stack, thereby removing this scenario from the stack. When this scenario has finished playing, the next scenario from the scenario stack is played. This is repeated until the scenario stack is empty. Since the scenarios are written in the interpreter language and are implemented as procedures, all we have to do is call the interpreter with the name of the scenario to be played. No extra administration is needed for playing the scenarios.

Because the sales strategy can change and the behavior of the client influences the sales talk, the Sales Engine must have ways of manipulating the scenario stack. For this, we have written several functions, which are summed up in Figure 5.6. With these functions the Sales Engine can play any scenario at any time.

| function | description |
|---|---|
| ScenarioInit | Initializes scenario stack from file 'scenario.dft'. |
| ScenarioReset | Empties scenario stack. |
| ScenarioIsEmpty | Boolean function to check if scenario stack is empty |
| ScenarioTop | Returns top scenario of scenario stack without removing this scenario from the stack. |
| ScenarioPopElement | Returns top scenario of scenario stack and removes this scenario from stack. |
| ScenarioPushElement | Puts given scenario on top of scenario stack. |
| ScenarioDeleteElement | Deletes a given scenario from scenario stack. |
| ScenarioInsertElement | Inserts a scenario in front of a given scenario. |

**Figure 5.6**: *Functions for manipulating the scenario stack.*

Since we want to minimize the amount of traffic on the network, we will store as many scenarios on the client's hard disk as the client will let us. But, just as objects, scenarios may become obsolete or may be altered by the provider. So for scenario's we have to store the following information for each scenario:

*Scenario version number.* The provider sends a list of all possible scenarios to the client. This list also contains the latest version numbers of his scenarios. All scenarios on the client's hard disk which are outdated are deleted.

*Date of last use.* This gives us the exact date on which the scenario was last executed. With this date, we can perform some sort of garbage collection. All scenarios not used for a certain amount of time are deleted.

The check for 'version number' and 'date of last use' will be done during the startup sequence of the K3 User Program.

## 5.4 Variable administration

The variable administration is used as a black board to exchange variables among different parts of the system. All parts of the system can store variables in the variable administration and all parts of the system can read the variables. The variables are used for very different purposes. The IPI uses them to analyze the client, the Sales Engine uses them to determine which scenario to play next. During initialization, the system reads the client profile and stores the variables in the variable administration. The IPI uses these variables to fill in the IPI–questionnaire, while newly acquired answers are stored in the client profile and the variable administration. The different rule bases in the system (e.g. in the Sales Engine and IBI) use the variable administration to set and store variables. Because these variables are accessible to all, the IBI can set a variable which is used by the Sales Engine to determine a new scenario. Figure 5.7 shows how the different parts of the system use the variable administration as a black board.

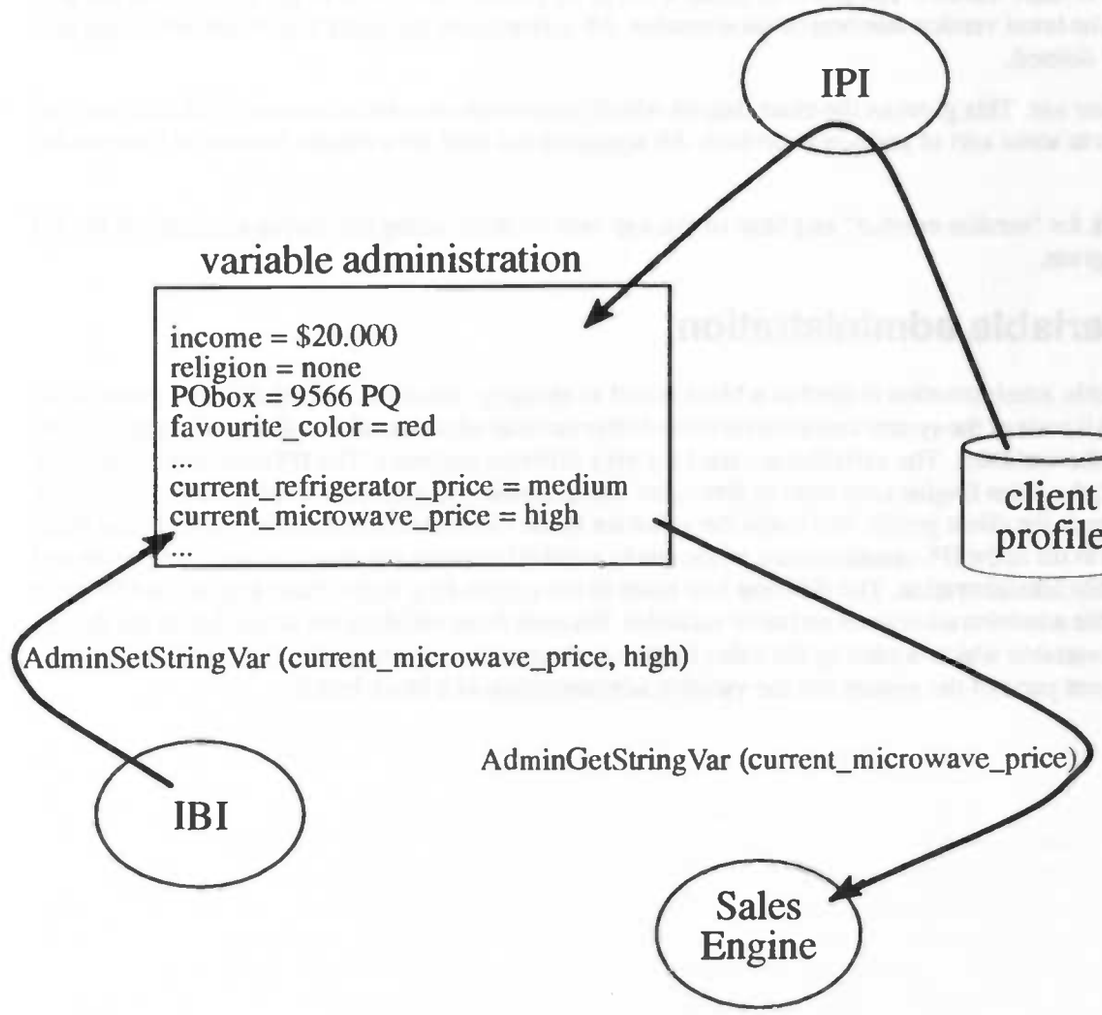



**Figure 5.7**: *The variable administration is used as a black board to exchange variables.*

A variable can be seen as a pair <*label*, *value*>. A variable can be accessed by its label to get its value. The label of a variable is a string denoting the name of the variable. The value of a variable can be either a string, an integer or a floating point number. So, if we want to create a new variable, we have to create a label and set the type of the variable to 'string', 'integer' or 'floating point' and then give the variable a value.

For manipulating the variable administration, we have created a number of functions, summed up in Figure 5.8. With these functions, we can add, delete, inspect and alter variables and their values.

| function | description |
|---|---|
| AdminVarExists | Check to see if variable exists. |
| AdminAddVar | Add a variable |
| AdminGetVarType | Get the type of the variable. |
| AdminSetFloatVar | Set value of floating point type variable. |
| AdminSetIntVar | Set value of integer type variable. |
| AdminSetStringVar | Set value of string type variable. |
| AdminGetFloatVar | Get value of floating point type variable. |
| AdminGetIntVar | Get value of integer type variable. |
| AdminGetStringVar | Get value of string type variable. |
| AdminDeleteVar | Delete a variable. |

**Figure 5.8**: *Functions for manipulating the variable administration.*

## 5.5 Sector overlay administration

In order for the K3 User Program to work, a sector overlay has to be sent to the client. Such a sector overlay contains sector and supplier specific information, as seen in chapter 4 and chapter 6. Since the sector overlay consists of files, which are processed by different parts of the system, and have standard names, all the sector overlay administration has to do is to see to it that the correct sector overlay is present on the disk of the client.

Each sector has its own sector overlay. But a supplier, not happy with the current sector overlay, can alter this sector overlay and so create his own version of the sector overlay. One of the goals of ITS MAGIC was to minimize the data exchange. So we store the sector overlay of the supplier on the local disk of the client, each with their own version number. If the client contacts a supplier, we first check to see if the sector overlay with the right version number is located on the hard disk of the client. If this is the case, we use this locally stored sector overlay. If the sector overlay is present, but the version number is outdated, we remove the sector overlay and send the new sector overlay to the client. If the sector overlay is not present, we send the sector overlay to the client.

Besides a version number, we also store a 'date of last use', which gives us the date on which the sector overlay was last used. When the client hasn't contacted the provider for a certain amount of time, the sector overlay is deleted.

## Chapter References

[1] D. Parnas, "On the criteria to be used in decomposing systems into modules", *Communications of the ACM, vol. 15, no. 12, 1972,, pp. 1053−1058.*

# Chapter 6 Sector Overlay Programming

## 6.1 Introduction

In this chapter, we will describe how the K3 User Program handles programs in the sector overlay. In section 6.2 we will see the interpreter that is used for this, and why we have chosen to use this specific interpreter. The interpreter is a 'stock' interpreter, taken from the Internet, so we needed to modify it for our purposes. Section 6.3 describes the way the interpreter has been modified for the K3 User Program. After this, the programming functionality that is available to the sector overlay programmer is described in section 6.4. Unfortunately, we did run into some problems with the interpreter during the implementation of our demonstration program. The problems and their solution can be found in section 6.5. The remainder of this introduction will be used to refresh the reader's memory by summarizing the reasons there are for both having sector overlays and having an interpreter.

### 6.1.1 Sector overlays

Sector overlays are an important feature of the ITS MAGIC architecture, as can be seen in Figure 6.1. Placed upon the K3 User Program, it provides the supplier with a platform which enables him to easily create a presentation of his products. For this, he only has to take an existing sector overlay for his line of business (a.k.a. sector) and configure it according to the way he likes to present himself. Besides this information, the sector overlay also needs other data from the supplier like information from his client database, logos and product descriptions. The way the client is approached by the virtual salesman is already contained in the sector overlay. The sector overlay will provide several ways to approach different customers, from which the supplier only has to pick one. When the supplier is not happy with existing sector overlays, he could either have one modified for him or create his own. Of course, this provides much more flexibility to the supplier, but it also means doing a lot more work.

We see that sector overlays are a way to make the K3 User Program a general–purpose program. Every sector will have its own sector overlay, and as time passes, we will see that more and more specialized sector overlays will be created. The ITS MAGIC architecture, with its sector overlays, ensures that there is a stable environment for these overlays.

To the user, the architecture also has its advantages. He will only need the K3 User Program for his tele–shopping, and not a different application for every vendor. Of course, this would not have been possible without having some form of overlay.

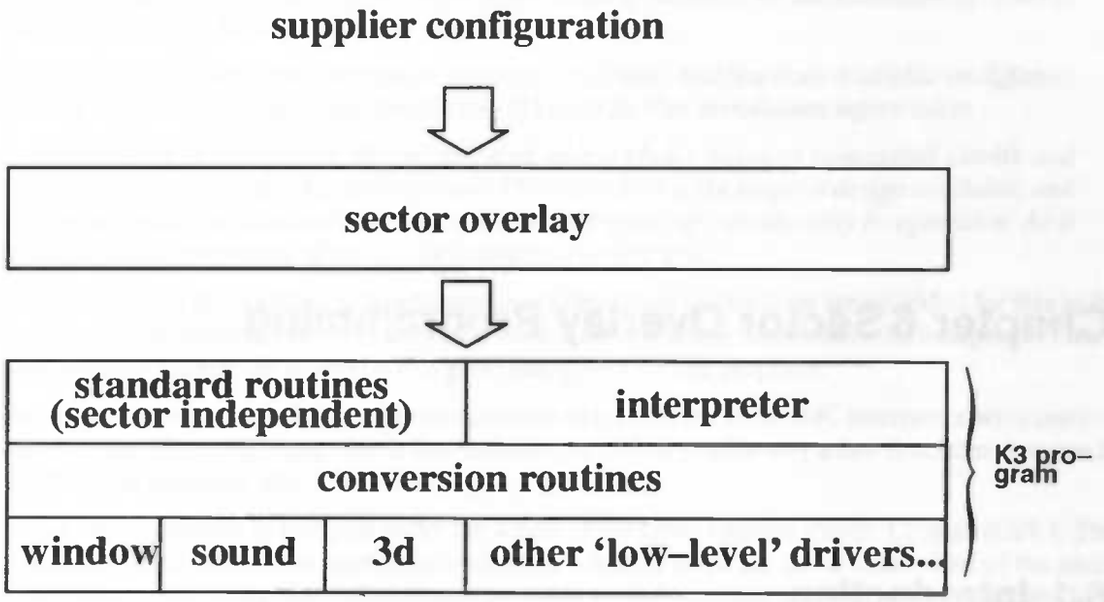



**Figure 6.1:** *The ITS MAGIC architecture*

### 6.1.2 Interpreter

Since we have created the demonstration program using Borland C, we could use a dynamic loadable library (or DLL) as the implementation of a sector overlay. Every time a new sector overlay is needed, it can be downloaded from the supplier and activated by the K3 User Program. However, this has one major drawback: it will only work on Windows 95 systems or compatible systems. On the moment, this will not pose any problem for most of the home computer owners, but given the speed with which both hardware and software keep developing, it may threaten the future of the system. We definitely do not want to update every sector overlay there is when a new operating system becomes the new standard. To prevent this, and to allow other platforms to have their version of the K3 User Program, we have to make the sector overlay platform independent. In other words, we have to interpret the contents of the sector overlay within the K3 User Program itself. For this, an interpreter is introduced.

## 6.2 Interpreter choice

In this section, we will discuss which interpreter we are going to use for the K3 User Program. First we will choose between making our own interpreter and using an existing one in section 6.2.1. After that, we will look at the demands the MAGIC architecture poses on the interpreter in section 6.2.2. Finally, section 6.2.3 will give a motivation for the choice that we have made.

### 6.2.1 Crafting or shopping?

Now that our need for an interpreter has been established, we have two options. The first option is writing the interpreter ourselves, the second option is to use an existing interpreter. We will discuss the advantages and disadvantages of both options, which are summarized in Figure 6.2. After this, our choice will be presented.

First, let's consider writing our own interpreter. The advantages are obvious: it will be a tailor–made solution for our needs. The interpreter will behave exactly the way we want it to behave, and it will provide exactly the functionality we require. Furthermore, since we have adaptability in mind, it will be written in such a way that modifications can easily be made.

However, choosing to write our own interpreter also has a drawback: we have to do it ourselves and it will take a lot of time to do so. A language has to be developed, specified and implemented. The implementation requires the usual interpreter elements [1]: scanner, parser and code execution.

The second option is to use an existing interpreter. The major advantage in this case, of course, is that we don't have to write it ourselves. However, it may be difficult to find an interpreter that fits our needs. Since it may not be obvious from the outside that an interpreter does not fit our needs, we may have to look closely at several interpreters before we find one that does. Finally, even if we succeed in finding a good interpreter, it will probably still be necessary to modify it to make it fit. In all cases, we have to find a way to add the K3 functions (see chapter 4) to the interpreter's functionality.

Taking all these advantages and disadvantages into consideration, we decided to use an existing interpreter for the demonstration version of the K3 User Program. Writing our own will take too much time and is beyond the scope of this project. However, at a later stage this interpreter could always be replaced with a better, more efficient, interpreter.

| *Writing our own interpreter* | | *Use an existing interpreter* | |
|---|---|---|---|
| + | *exact match of required functionality*<br>*easy to modify* | + | *we don't have to do it ourselves*<br>*saves time* |
| – | *we have to do it ourselves*<br>*takes a lot of time* | – | *maybe hard to find*<br>*modifications certainly needed* |

**Figure 6.2:** *Advantages (+) and disadvantages (–) of both writing our own interpreter and using an existing one*

### 6.2.2 Demands

The MAGIC architecture implies certain demands for the interpreter. Since we choose to use an existing interpreter, we must know these demands before we can select a good interpreter. In this section, we will describe these demands.

First of all, the interpreter should be written in C, since this is the language we are using to implement the demonstration version of the K3 User Program.

Second, the interpreter source should be easy to read. Only when this condition is met, will it be possible to make the necessary modifications. Modifications include, but are very probably not limited to, adding the K3 functions.

Third, the interpreter should have very small execution steps. Preferably, there should be a function that can be called to perform such a small execution step. In this way, we can let the current program fragment run amongst other tasks. For instance, consider a refrigerator door that is opened by a sector overlay program fragment. While the door is opening, it should still be possible to do other things like making the dog walk around the kitchen. When a sector overlay routine is performed in whole, before control is being returned, this would not be possible.

### 6.2.3 Our choice: bwBASIC

Using the search facilities nowadays offered by the Internet, we were able to find only a few freely available interpreters. Of these, most were not written in C. Of the two interpreters that were written in C, only one had easy–to–read source code: the bwBASIC interpreter, version 2.10.

The following is an excerpt from the documentation file that came with the package:

> *The Bywater BASIC Interpreter (bwBASIC) implements a large superset of the ANSI Standard for Minimal BASIC (X3.60–1978) and a significant subset of the ANSI Standard for Full BASIC*

*(X3.113–1987) in C. It also offers shell programming facilities as an extension of BASIC. bwBASIC seeks to be as portable as possible.*

*bwBASIC can be configured to emulate features, commands, and functions available on different types of BASIC interpreters; see the file INSTALL for further installation information.*

*The interpreter is fairly slow. Whenever faced with a choice between conceptual clarity and speed, I have consistently chosen the former. The interpreter is the simplest design available, and utilizes no system of intermediate code, which would speed up considerably its operation. As it is, each line is interpreted afresh as the interpreter comes to it.*

bwBASIC can easily be modified to handle new functions. Sample functions are provided for this in the source code, and the documentation exactly tells what to do to add new functions to the interpreter. Therefore, although the interpreter is slow, it is a good interpreter for our purposes.

At a later stage, however, it will probably be desirable to replace the bwBASIC interpreter with another, faster interpreter. Since the interpreter is implemented as a library, with only a few functions that can be used by the main program, this will be a fairly easy job.

The bwBASIC interpreter is released under the terms of the GNU General Public License (GPL). This means that any derivative of the interpreter should be released under the same terms. One of the terms is that the source code of the derivative should be made available to anyone who wishes to obtain it. In our case, this would mean the modifications to the interpreter have to be made public, including calls to the conversion routines that implement the K3 functions. Although in itself this will not be a problem, not having the full rights to all parts of the K3 User Program might not be desirable. Therefore, this could also be a reason to replace the bwBASIC interpreter eventually.

## 6.3 Modifications to the interpreter

As we have seen before, the interpreter will need several modifications to fit our needs. The following modifications were made:

- General modifications to make the original interpreter work.
- Modifications to forbid certain BASIC commands that should not be used in a sector overlay.
- Functionality was added to have the interpreter perform a single execution step. This is used by the K3 User Program to call the interpreter at regular intervals using a timer.
- The K3 variable types were added to the interpreter by means of a variable table.
- The K3 functions were added.

A detailed description of these modifications are given in the remainder of this section.

### 6.3.1 General modifications

Several modifications were necessary to make the original bwBASIC interpreter work on our development platform. However, most of these changes are trivial, since they only involve configuration options in the header files. Consequently, these changes will not be discussed here.

One modification that is worth mentioning is the addition of several NULL–checks. In some cases, not checking for variables that have their value set to NULL leads to a system crash. Since there are still unexplained crashes (see section 6.5), it is likely that not all these problems are found and corrected.

### 6.3.2 Removal of unwanted functionality

Some commands offered by bwBASIC should not be used in sector overlays. The most important example of this is the option bwBASIC offers to start a shell and execute a system command in it. Imagine a pro-

grammer who places a command in his sector overlay that would format the user's hard disk. Needless to say, this kind of 'functionality' should not be part of the K3 User Program.

Figure 6.3 gives a list of all bwBASIC commands that are removed together with the reason for doing so.

| *bwBASIC command(s)* | *Reason for removing it* |
|---|---|
| *chain / common / merge* | *These are commands for passing control to other BASIC programs, and this is not allowed.* |
| *chdir / close / eof / field / get / input / kill / line input / loc / lof / mkdir / name / open / put / rmdir / width / write* | *These are commands for doing file i/o. For safety reasons, file operations are only permitted through K3 functions* |
| *any non–basic command* | *Non–basic commands are executed as system commands, i.e. they are executed in an operating system shell. In this way, the user's computer can be accessed without any control. Of course, this should not be allowed.* |
| *delete / do num / do unnum / edit / list / load / run / save / troff / tron* | *These are editing commands. No program editing is necessary in a sector overlay.* |
| *cls / inkey$ / input / locate / pos / print* | *These are screen i/o routines. As there is no text screen in the K3 User Program, these routines cannot be used. K3 functions are provided for this functionality,* |
| *environ / environ$* | *These are routines to get and set environment strings. As these are beyond the control of the K3 User Program and can give information about or change the behavior of the user's system, they cannot be allowed.* |

**Figure 6.3**: *bwBASIC commands that are removed*

### 6.3.3 Timer preparation

As we mentioned earlier, we want the interpreter to run on a timer, i.e. the interpreter should perform small execution steps at regular intervals. This enables the K3 User Program to do other tasks simultaneously, thus creating a multi–tasking environment.

The bwBASIC interpreter was already prepared for scheduling: it has a function that performs only a single, small execution step, after which control is returned to the caller. For this function, a new interface function is written that does a single step, and as a result it returns the status of the execution stack (empty or not empty).

Whenever a bwBASIC subroutine is called by the K3 User Program, it is placed on the execution stack, after which the timer takes care of executing it. We must take care not to be running two bwBASIC subroutines at the same time, as there is only one execution stack[2]. Therefore, we need to check the execution stack before we start a subroutine: we can start subroutines only if the execution stack is empty. Failing to meet this requirement can lead to a crash, as we have experienced several times.

2. This is not exactly true. The bwBASIC interpreter was written to be able to handle more than one BASIC program running at the same time. However, we could not get this to work, and it seems that the multi–tasking capabilities of bwBASIC are a work in progress.

### 6.3.4 K3 variable types

The bwBASIC interpreter handles *number* and *string* variables. This is a problem, because the sector overlay programmer should also be enabled to use the K3 variable types like **K3Real**, **K3Coord** and **K3Object**. However, adding these types as fundamental BASIC types would require a lot of changes to the interpreter.

Therefore, we came up with another solution: we added a variable table. This table holds the K3 variables, and the sector overlay programmer uses the indices, which are numbers, in the sector overlay. The use of this table is completely transparent to the sector overlay programmer, except for one thing: he has to remove the variables he doesn't need any longer. If he fails to do this, the table can become completely filled.

Figure 6.4 depicts how the table works. In this case, the statement **glass = K3ReadObject ("glass.rwx")** is executed by the interpreter. First, a free slot in the variable table needs to be found. Here, the slot with index 4 is found to be empty. Then, the object is read and the resulting K3Object is stored on the heap. The variable table contains a pointer to this object. Lastly, the index is returned as a result of the BASIC function. Now, the value of the BASIC variable 'glass' is 4. This variable can be used to access the K3Object, just like a real C pointer.

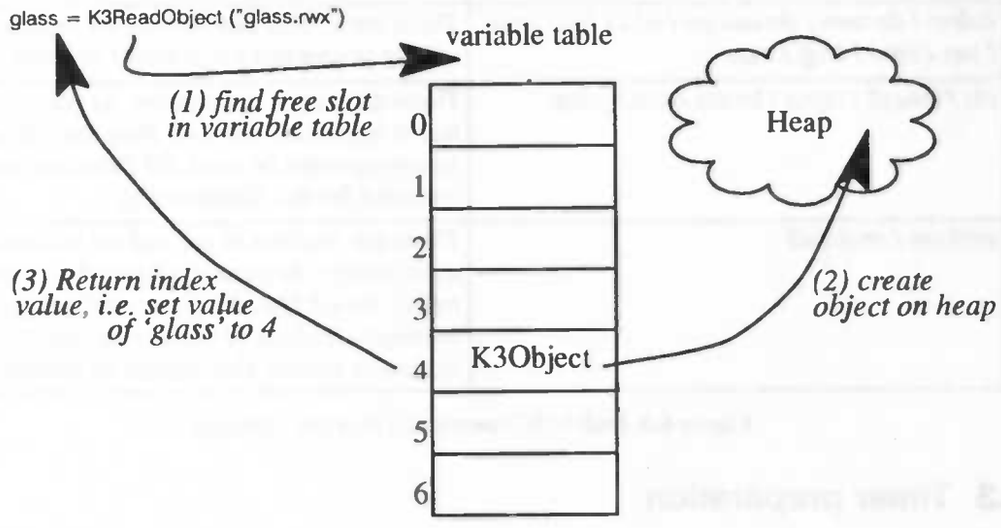



**Figure 6.4**: *The variable table for the K3 variable types*

In some cases, the sector overlay programmer does not have to remove variables from the table himself. The **K3DestroyObject** function, for instance, takes care of both removing the object from the heap and freeing the table slot it occupied. In other cases, for instance when a K3Coord is needed to temporarily hold some object's position, the programmer does have to remove the variable from the table himself. This is needed because the interpreter has no way of knowing if the variables in the table are still needed; there is no garbage collection for this.

### 6.3.5 K3 functions

The most prominent modification to the bwBASIC interpreter is the addition of new functions: the K3 functions. This addition changes bwBASIC into a sophisticated multi-media programming language, with provisions for 3D, sound and window manipulation.

The K3 functions are interfaces to the corresponding C functions. Where necessary, the variable table as described in section 6.3.4 is used to store function results that cannot be stored in BASIC numbers or strings.

The addition of the K3 functions was not very difficult: the bwBASIC documentation describes in detail how functions should be added. For more information about adding new functions, we refer the reader to the K3 User Program *Implementator's Guide* [2].

## 6.4 Overlay functionality

The sector overlays can now be programmed using bwBASIC and the K3 functions. The K3 functions are listed in the K3 *Function Reference* [3]. In bwBASIC, all usual programming language elements are available. The K3 calls can be used just like any other BASIC function. For example, to display an information window with a text and a window title, one can add the following line in the BASIC part of the sector overlay:

result = K3InfoWindow ("This is the message", "window title")

In this case, the variable *result* contains the exit code for this function, as described in the K3 Function Reference [3]. For more information about programming in ANSI BASIC, we refer the reader to all the excellent literature available on this topic, for example

## 6.5 Encountered problems

While working on and working with the bwBASIC interpreter, we had to deal with several problems. Some of these problems were solved almost as soon as they were discovered, others still are present at the time of writing.

One of the first serious problems was with the **K3RotateObjectAround** function, which has too many parameters for the original bwBASIC configuration. As the number of parameters that is given for a function is not checked against the maximal number of parameters, this problem was not detected at first. Only when the interpreter started to behave unpredictably did we notice that there was something wrong. Finding the error was a problem in itself, and it took quite some time before we found out that we only had to change a constant in a header file.

The most mysterious problem still exists at the time of writing this document. Sometimes, the entire program crashes with a certain sector overlay program. This problem goes away when a number of empty lines are added to the program file. Hours of time were spent trying to find this problem, but until now all without success. We are still without a clue on this one, so sector overlay programmers be warned: whenever the K3 User Program crashes and you don't know what's going on, try adding a few empty lines here and there. And don't ask why.

## Chapter References

[1] Charles N. Fischer and Richard J. LeBlanc Jr., "Crafting a Compiler", *Benjamin/Cummings, 1988.*

[2] R. Buist, J. de Graaf and W. Wichers, "K3 User Program Implementator's Guide", *University of Groningen, August 1996.*

[3] R. Buist, J. de Graaf and W. Wichers, "The K3 Function Reference", *University of Groningen, August 1996.*

[4] J.G. Kemeny and T.E. Kurtz, "Structured BASIC Programming", *Wiley, New York, 1987.*

# Chapter 7 Path planning and natural motion

## 7.1 Introduction

In the K3 User Program the client using this program will be represented as a 3D human being in a 3D virtual world. This 3D human being is called the ego object (see also chapter 5). At this moment the client can control the ego object by using the mouse. Two kinds of manipulation of the ego object are provided. Firstly, the client can move forward and backward by moving the mouse up and down respectively while pressing the left mouse button. He is also able to turn the ego object around by moving the mouse button to the left or right (for a left or right turn resp.) while also pressing the left mouse button. This kind of manipulation will be called direct manipulation. Secondly, the client can also move the object to a position in the virtual world by just clicking at this position with the right mouse button. The ego object will then automatically walk to this position. This action sequence will be called telepathic manipulation (The user wants to go to a position and the object automatically moves towards this position).

By using a 3D virtual world (also called the scene) we try to use the world metaphor to make the client feel confident with the system. He can interact in a very simple way with the system, because he acts in a natural environment. To view a product of the supplier he just clicks the product and the ego object will walk to the product. After the product has been clicked, the supplier is able to demonstrate this product. For instance, the client clicked a refrigerator in a kitchen application. The kitchen supplier can make his own demonstration of this particular refrigerator, which opens the door and shows the nice inside of the apparatus etc.. When we want to make the whole experience as natural as possible, the ego object has to move in the scene like a real human being. If there are other moving objects in the scene, like other human beings or pets, they should also move in a natural way. The moving objects must also avoid obstacles in the scene in a natural manner.

So there are two kinds of manipulation. On the one hand, when the ego object is manipulated directly, atomic movements are executed one by one. Such an atomic movement (Eg. a step forward) consists of a standard motion set of the (movable) parts of the ego object. If the client manipulates the ego object directly, the system reacts by executing the standard atomic motion set, belonging to client input, if the position the ego object wants to move to, is obstacle–free.

On the other hand, when the ego object is manipulated telepathically, the movement of the ego object in the scene is composed of two components. The first component is the movement of the entire object from a start position to a goal position. This movement has to be of a natural shape and should not just contain sharp corners and straight lines (see also Figure 7.1). The other component concerns with the motions of the parts of the ego object. When the ego object represents a human being, the parts of the ego object have to move like a real human being. In contrast to the direct manipulation, several actions have to be performed when the ego object is manipulated telepathically. This chapter concentrates on the problem of the real–time motion execution of the ego object from a start position to a "clicked" goal position.

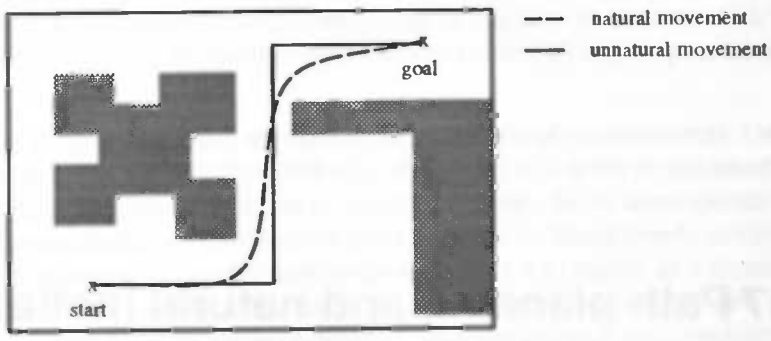



**Figure 7.1**: *2D representation of a kitchen floor plan. The grey areas are objects in the kitchen. Moving in the kitchen has to be natural*

After this introduction several combinations of the execution of natural motion will be described. In the subsequent section a lot of solutions to the planning problem will be given. The last section will tell about the methods, concerning the path planning problem and natural motion, used in the K3 User Program.

In this chapter a lot of solutions, mainly to the path planning problem, will be named, but not completely described. Maybe some of these solutions don't seem to be interesting at this moment, but they could be useful in the future.

## 7.2 Real–time natural motion

The goal of the research described in this chapter is to build a system that can execute real–time (natural) motion. The desired system would fulfill commands to move to a desired (goal) position while avoiding obstacles in the environment. To execute a motion task the ego object must combine the ability to plan motions and to execute them. Three different kind of combinations can be found:

- plan a global path from start to goal position. Use this global path to plan all the needed motions to move (in a natural manner) from a start to a goal position. When all the planning tasks are completed, the execution of the planned motions begins. This is a sequential process (see also Figure 7.2 a)).
- plan a global path from start to goal position. Instead of planning all the motions we now plan and execute the motions one by one. During the execution the environment will continually be checked for unknown or moving objects crossing the planned path. When such a crossing occurs, two possibilities exist to adapt the global path. The first possibility determines a new start position and plans a "remaining" global path from the new start position to the goal. Another possibility solves the crossing of the unknown or moving object by executing a local path planning routine. The global path will change locally if problems occur. A useful technique is to treat a path as a flexible entity, also called elastic bands ([12]). The process described here is a feedback process, which is more time–consuming then the first described sequential process, but is certain to be a lot safer in dynamic environments (see also Figure 7.2 b)).
- plan no path at all and just plan and execute the motions one by one, using local information and keeping the goal position in mind (see also Figure 7.2 c)).

The first and second combination are covered by inverse kinematics. The motion is planned by using determined positions to move to. The determination of the positions to move to is called path planning. The problems and several methods considering path planning will be given in the next section. The last combination is covered by kinematics. Here a motion is performed and the results of this motion causes the

determination of the next motion. The path that will be followed arises by the execution of the determined motions.

We don't want to use this last combination at this moment. This combination may be useful for direct manipulation. When used for telepathic manipulation it can't be guaranteed that the goal will ever be reached. The situation can occur, that the ego object will keep moving in the scene, trying to reach the goal, which may never happen. Also the way we are able to plan motions (later more), cannot be used directly in combination with the kinematics variant. In literature the following was found, which may be used in the future, concerning the last combination. Using the kinematic approach, the search is performed while moving, and depends only on local information. Q_learning based approaches ([3]) or the use of oscillatory networks to find out the optimal path ([4]) are examples among other possibilies. In [13] a simple model of action sequences and active vision grounded in a connectionist architecture is presented. The model adapts its behavior on–line thanks to a reinforcement learning which, in response to a feedback both "weakly informative" and delayed, modifies the weights according to a Temporal Difference (TD) strategy ([15]). The simplest way to implement TD in a system is using the Q–learning algorithm ([14], [15] and [16]).

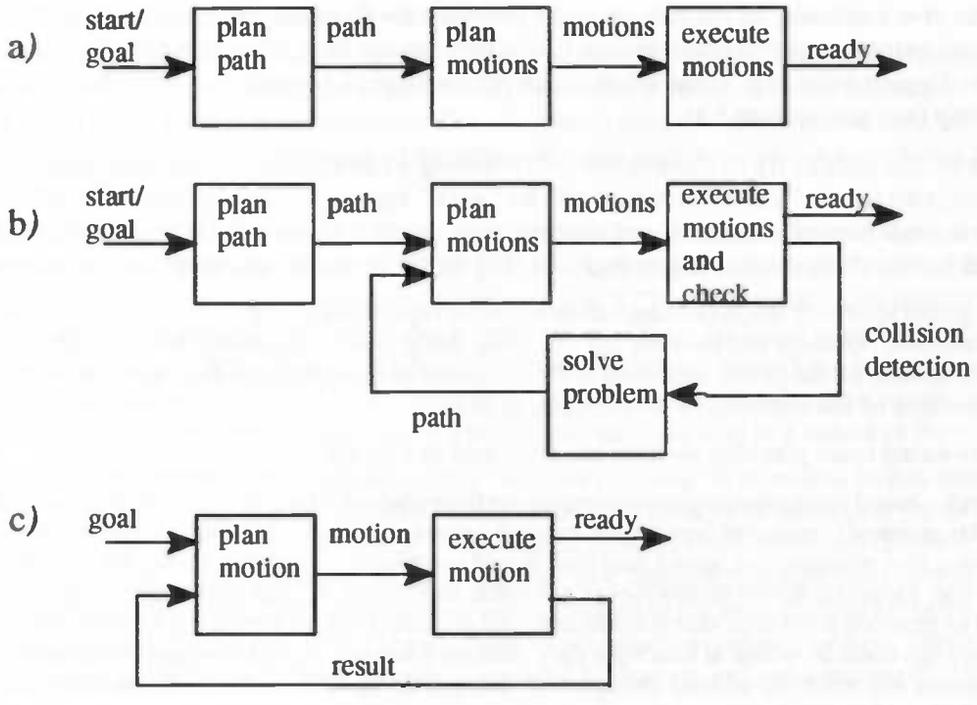



**Figure 7.2:** *The three planning and execution combinations*

## 7.3 Path planning

The objective of path planning is to automatically guide an object in a workspace with obstacles. In this section a short summary of several solutions to the path planning problem will be given. This summary can be given due to a brief study of literature concerning this topic. The articles used for this study can be found in the Chapter References at the end of this chapter.

Most of the articles refer to solutions to the path planning problem for mobile autonomous robots. The primary drawback of most path planning methods for robots is that they operate in known environments. In these environments, a map of a building or factory can be readily obtained and used to perform path planning for a mobile robot. However it is impossible to guarantee that the map is accurate, since it does

not record the presence of people or obstacles such as boxes left in passage ways, which leads to the conclusion that completely known environments rarely exist in practice. In our case, a virtual world, we can guarantee that the map is accurate. In a virtual world we can control all objects present, including the moving ones.

Keeping the K3 User Program in mind, we can define two kinds of environments. On the one hand, we can define the **variable** environment. This kind of environment will never be the same. One time, for example, it will be a kitchen, the other time it can be a shopping center. So no assumptions can be made while dealing with a variable environment. On the other hand, we have the **fixed/steady** environment. This environment almost never changes. When the environment starts, for example, as a factory, it will remain a factory. Details in this factory lay–out can change, like the introduction of some new equipment, but the environment remains a factory. Both kinds of environments can occur in two conditions, namely a static and dynamic condition. When an environment is in a **static** condition, nothing changes during the determination and execution of a valid path from a start to goal position, in other words there are no moving objects. When dealing with **a dynamic** condition, several moving objects can appear.

Some articles give a solution for the path planning problems for ***fixed/steady*** environments. These solutions use a very extensive precomputation step, before they can use their, often fast, path planning method (for example slippery cells [11]). These solutions are not useful to us, because our virtual world will rarely be ***fixed/steady*** (see also section 3.4).

In most articles the authors try to classify the path planning methods into several categories. The most frequent categories are the "global" approach and the "local" approach to path planning. In other articles a difference is made between "search based systems" and "reactive planners". These are almost the same as the global and local approaches respectively. In [10] the following is said about the global approach:

> "The global approach involves constructing a concise representation of the connectivity graph of admissible positions and heuristically searching this graph for a globally optimal path. ..... The drawback of the global approach is the extensive precomputation step involved in the construction of the connectivity graph. ([10], p. 787)"

In [9] search–based route planning systems are described as follows:

> "Search–based route planning systems represent their planning domain in a graphic form with weights assigned to each link based upon an application specific criteria function. For example, distance, fuel consumption, speed, and time would combine in a criteria function for planning a car trip. Heuristic search techniques (e.g. branch and bound, $A^*$, hill climbing) are then applied to produce a solution that is optimal to the criteria function attributes. Search–based systems are static in nature in that once they produce a weighted graphic search space, no alternations are made to address unexpected changes or updates in the environment. ..... search–based systems grow exponentially in processing time as the search space expands, translating into non real–time performance. ([9], p. 221)"

These (global and search–based) approaches to the path planning problem perform a search in a large solution space representing all the possible sequences of elementary moves in a discretized space including an explicit representation of the obstacles. This search, undertaken off–line and in a sequential way, generally aims to minimize a global criterion. The use of the $A^*$ algorithm is one possibility ([2]). However, these approaches are easily subject to combinatorial explosion. They also present weak adaptivity, so they are often used in ***fixed/steady environments.***

Descriptions of the other approaches can also be found in [10] and [9], for respectively the local approach and the reactive planning approach.

> "The local approach involves gathering partial information about the geometry of the configuration space which is used to generate heuristics which guides the search. The configuration space is a representational tool in which the object/workspace problem is transformed into that of a point object in an appropriately dimensional space (the configuration space of the object). The local approach requires no expensive precomputation step and runs faster than the global

> approach but requires powerful heuristics to guide the search. Such heuristics have been known to guide the search into dead–ends. ([10], p. 787)"
>
> "Reactive planners model their environment in a matrix representation utilizing an energy function that corresponds to the search–based criteria function. The search process, however, is in the form of a linked progression where only the next several moves are planned instead of the entire route. This allows the system to dynamically address changes in the environment by efficiently updating search space nodes that are effected. Reactive systems thus trade–off route optimality with processing time limitations in processing towards their goals and responding to obstacles in their path. ([9], p. 221)"

In literature there have been various local obstacle avoidance procedures proposed. A very popular alternative amounts to the computation of a potential field repelling on each obstacle and attracting at the destination of the moving (ego) object. Potential fields were pioneered by Khatib ([17]). The potential field approach to path planning involves developing a distributed representation of the workspace by computing potential fields over the configuration space. In a potential field, a potential 'wave', propagated away from a given goal point, determining for example the path length from a point on the wave to the goal, is stored. Other information, like obstacle positions, can also be stored in the potential field, in which case the potential field is called an *obstacle transform*.

Stochastic search techniques use the potential fields to guide the search. The main drawback of this approach is the extensive computation in developing the potential fields. Another problem of using the potential fields is the local minima problem. The distributed nature of the potential field approach makes it easily parallelizable. The complexity of the approach scales with the workspace size rather than the number of obstacles.

Another approach to local obstacle avoidance is heuristic navigation. Heuristic navigation as studied by Boreinstein et. al. ([18]) and Chattergy ([19]) guide the ego object to the goal by using rules to decide, based on local information, which path of those available is "best". Heuristic path planners can solve a wide variety of path planning problems, but a problem can always be found where a particular heuristic strategy fails. The appealing feature of heuristic obstacle avoidance procedures is that they require little processing and can produce rapid "reactive" behaviors. However, in these approaches since no planning is done the resulting execution paths may not be optimal and may lead to a failure of the mission ([12]).

In [12] a local obstacle avoidance scheme called "reactive planning" is presented. In this scheme the "path transform" is used, which was developed by A. Zelinsky ([20]), one of the authors of the article. The path transform is a grid based planning method. This method considers both minimum distance to a goal and collision avoidance. The path transform can be regarded as a "numeric potential field" which has the desirable properties of potential field path planners without suffering the local minima problem. The reactive planning approach yields a fast, efficient and robust local obstacle avoidance method. There has been other work which is similar to the path transform reported by Barraquand and Latombe ([21]).

The approaches to the path planning problem described so far are not part of the researches of the connectionist community. But there are also some connectionist approaches like the family of methods using the Kohonen's self–organizing topologic map idea to search an optimal path in a constrained environment ([5] and [7]). Elastic nets ([6]) are a famous member of this family. In [7] the following is said about using neural networks concerning the path planning problem:

> Neural networks present a computational paradigm for constructing collision–free path planning. Two of the most dominant motivations of using artificial neural networks are that they learn to classify their input or training patterns and that they are capable to make generalizations in situations not covered by the training set. A reinforcement scheme (see for instance [22]) is often used for obstacle avoidance learning ([24] and [23]). These approaches perform the search on–line and rely on local qualitative information (punishment/awarding). If we consider real robots, the most serious problem in reinforcement learning is a slow convergence of learning. Park and Lee ([25]) have developed a path planning algorithm by utilizing neural computations to minimize the collision penalty function. However, they have assumed that the locations of obstacles are known a priori.

Neural networks are also used to implement the potential field approach. In [10] a neural network architecture (WENN: Wave Expansion Neural Network) is presented that is specifically designed to implement potential field operations useful for path planning. [9] also presents a neural network approach to reactive route planning based upon the development of an application specific energy function in the Hopfield model. The energy function can combine various planning elements, like distance to the goal, obstacles to be avoided etc.. The presented system uses a digital terrain map as input.

In [8] a neural network system is presented that is able to do hierarchically trajectory planning by combining simpler trajectories to more complex trajectories via subgoals. The gradient network should allow on–line retraining for changing environments. The problem with the current approach is the quality and the training time of the classification function. Another problem is that there is just one attempt with the neural subgoal generator. If this fails, other planning mechanisms have to be used.

The advantage of the most connectionist approaches is the ability to improve their performance during their lifetime. They are also often adaptable to new situations and environments, because they are capable of making generalizations. A disadvantage is the need for training. The network can learn an effective collision avoidance behavior on the basis of acquired examples as well as from experiences during its lifetime. Using these approaches for an autonomous robot, which have be installed once in a new factory lay–out, the training time is no problem. In this situation the robot has to be trained maybe once in a few years. When the robot improves its behavior during its lifetime, it may act a little bit clumsy during the first times it meets new situations. After this (for a factory a short) period, the robot will act normal and will be able to avoid formerly unknown obstacles in a proper way. Our situation, a virtual world, is a bit different and will be discussed in the next section.

## 7.4 The methods used in K3

The main difference between the real world and our virtual world is that we can control our virtual world fully. In the virtual world we know the exact size and position of all the objects. In the K3 User Program a global object map is continually updated. When an object is added to the scene, is deleted from the scene, changes position in the scene or is manipulated in any other way, this will be processed directly in the object map. The object map is a grid representation of the scene. Every pixel of the grid contains a number, which represents the number of objects occupying each pixel. When an object is manipulated in the scene, first all numbers of the pixels which are occupied by the old object bounding box will be decremented by one. Next all numbers of the pixels which are occupied by the new object bounding box are incremented by one. The resolution of the grid is important for the precision of the world representation. A low resolution results in a rough representation of the objects in the scene. Object representations will become bigger, resulting in the connection of objects that are in fact not connected. So the number of potential paths will decrease and the path planning problem will become more difficult and maybe even unsolvable. A high resolution results in a perfect representation of the objects in the scene. On the other hand a lot of memory will be needed to store the high resolution map. Also the updating of the object map will become much slower. So a balance between the number of pixels and the wanted precision has to be found. We used one pixel for every 100 $cm^2$ of the environment, but this can be changed for every particular situation.

The object map is coupled to the current ego object. The height of the ego object is used to build an object map which only contains objects that are placed low enough to form an obstacle for the ego object (i.e. the ego object cannot pass underneath them). So when changing the ego object to a taller or smaller one, a new object map has to be made.

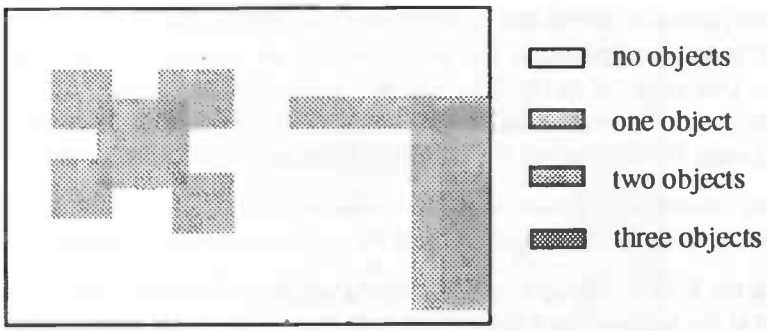



**Figure 7.3**: *An example object map*

The K3 User Program will meet a lot of different environments. Due to the adaptation of the program to the client, the possibility exists that each client will get a different environment. For example, a client wants to buy a kitchen. This kitchen will be presented in his own house. The positions of the sink unit, refrigerators, cupboards etc. will be adapted to the taste and wishes of this client. Because a lot of different houses and clients exists the number of different environments is infinite. And of course, each specific sector (or even supplier) uses its own environments. So the K3 User Program doesn't deal with a *fixed/steady* environment, but every execution of the program will often result in a completely new environment.

Because the environment isn't *fixed/steady* we can't use methods which are based on a very extensive precomputation step. If we should use such methods, this extensive step has to be done for almost every execution of the K3 User Program, which will result in long waits that are a nuisance to the client. For the same reason a lot of the neural network approaches found are also unusable, because there is no time to learn or (self–)organize the network.

To simplify the problem we assumed the environments we use are static. The positions of all the objects are known. When there are any moving objects in the scene, this assumption can result in collisions between these moving objects. Such collisions look silly, but have no other consequences, because we are living in a virtual world. A first improvement, in the future, will be halting all the moving objects during the planning and the execution of the motions of all these objects, except for the moving ego object.

An important demand on the motion planning system, is that the motions have to look natural. Hans Seinhorst and Eric Grave have done research in this area. They developed the motion editor and composer *MedCom* [26]. In cooperation with Hans Seinhorst, we have made a beginning with the natural motion of human objects in the K3 User Program. The routines, developed by Hans Seinhorst, depend on anchors to plan a path. Such an anchor contains the following information:

- position: This is the position of the anchor in the (3D) virtual world.
- curve: The shape of the curve (Eg wide or sharp) that will be used to pass the anchor.
- priority: The priority of passing the curve. Using a high priority the route will pass the exact anchor–position. A low–priority anchor is used to indicate the direction of the path; the planned path may not pass directly over the anchor position.
- motion: The kind of motion (Eg walking, jumping, running) the object has to perform between this and the next anchor.
- emotion: The kind of emotion (Eg happy, sad, anger) the object has to assume between this and the next anchor.
- speed: The object has to move with this speed between this and the next anchor.

The first three elements (position, curve and priority) are used to plan the global path of the entire object moving from the start to the goal position. The determination of a natural shaped path can be solved in several different ways. One solution can be the calculation of Bezier–curves through (?) the anchors. But if the position of an anchor is far away from the positions of his neighbor anchors, the Bezier–curve will not pass this anchor. Using Bezier–curves the path will become too smooth (fluently).

Another method draws circles near the anchors. The natural path will be formed by connecting the tangents of two neighboring circles. The tangents cause the smooth transition between the circles.

This method is used in the K3 User Program. The method will be called with a list of anchors and a lookdirection of the object at the beginning of the natural path and a direction the object has to face at the end of the path. If necessary, two extra anchors will be inserted to guarantee that both lookdirections can be managed. By using the position of the previous and next anchor, a circle will be placed around an anchor. The placement of the circle depends on the priority and curve of the anchor. The circle can take an inner or outer curve or even cross the anchor. When the circle is placed, a connection with the previous circle will be made by determining the tangents of both circles. This process will continue until all anchors are visited. Next the complete path will be checked for good placements of the circles and if some circles are badly placed, this will be corrected.

The global path has been calculated, so the movements of the object can be calculated. The stepsize of the object during a curve is dependent on the size of the circle. The bigger the circle, the bigger the stepsize. At the straight parts of the global path a constant speed will be used. In the future the motion of the limbs of the objects will also be planned. These motions can be planned by using *MedCom* [26]. This motion composer uses the model (e.g. human being, dog or spider) of the object and the information given by the last three elements (motion, emotion and speed) of an anchor, to plan natural motions of the limbs in different conditions.

With the aim on using the routines of Hans Seinhorst, we have chosen for inverse kinematics. We determine the motions (of the parts) of the ego object by using calculated positions in the scene. So we had to find a method which delivers the needed positions in the virtual world. To save (valuable) time we didn't want to develop a complete perfect path planning system by ourselves. The articles we read offer a lot of ideas, but no working systems. Because we have already translated the scene into a grid representation, in which the objects are stored, we thought a potential field–like approach would be easy to implement. We could use the same grid representation for the storage of the potential field. Before we started programming, we started a search for available source code on the Internet. We were able to find a freely available path planner using potential fields: the Randomized Path Planner (RPP). This path planner was developed by Jerome Barraquand and Jean–Claude Latombe in the Robotics Laboratory of the Computer Science Department at the Stanford University. Randomized Path Planner was developed to solve path planning problems in high dimensional configuration space for robots with a possible high degree of freedom. This can be useful in the future, but we only used their algorithm to build a potential field by using the information of our object map.

The potential field used in RPP is a numeric potential field using a grid representation. The constructing of this potential field is done in three steps. Firstly, an "obstacle transform" of the free space cells is computed, from which a *distance skeleton* is extracted. This skeleton represents a digitized Voronoi diagram[3] of the free space in the environment. Joining the highest values in the obstacle transform with line segments will yield a distance skeleton. Then the goal cell is connected to the distance skeleton. Secondly, a distance transform from the goal cell to all cells that are member cells of the distance skeleton is computed. Thirdly, another distance transform from the member cells of the distance skeleton to all the remaining free space cells in the environment is computed. The end result is a "numeric" Voronoi diagram of the environment. Since this method maximizes clearance from obstacles it can divert the solution (too) far from the shortest path ([12], p.451).

3. The Voronoi Diagram of a finite set S of points in the plane is a partition so that each region is a set of points that are closer to a certain point in S than to any other point in S.

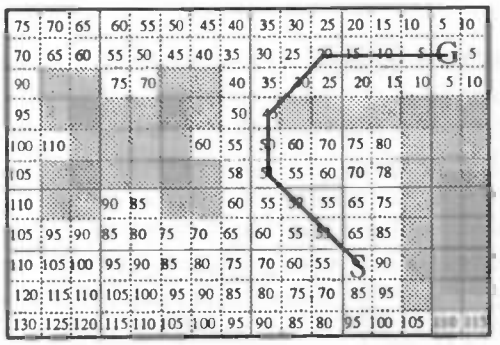



**Figure 7.4**: *An example of a potential field, without local minima*

We used the potential field functions offered by the RPP code of Barraquand and Latombe. The ultimate path to the goal is traced by following the path of steepest descent (see Figure 7.4). If no downhill path exists then we can conclude that the goal is unreachable or the potential field contains local minima. As expected the implemented potential field didn't calculate the shortest path (see Figure 7.5). During the test phase of the implementation, it became obvious that the route as implemented by RPP would look very silly. Therefore, we decided to optimize the potential field algorithm for our problem in a 2D environment.

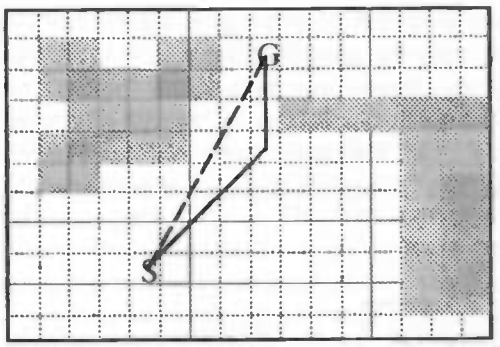



**Figure 7.5**: *The Voronoi–based route as implemented by RPP vs. the desired route*

The source code of the used routines of the Randomized Path Planner doesn't contain useful comments, which makes changing the code very unattractive. Instead of calculating an entire updated potential field, the improvement of the algorithm is solved locally. The process starts at the initial start position. To each potential field value of all neighbor pixels of the start pixel, an extra value is added. This value represents the amount of divergence with respect to the direction to the goal. The value of a particular neighbor pixel can be calculated by the determination of the angle between the vector to the goal and the vector to the neighbor pixel, both starting from the start pixel (see Figure 7.6). The neighbor pixel with the smallest total value will become the new start pixel and the process starts again. So, the energy function of the potential field is influenced by the following criteria:

- distance to the goal
- direction to the goal
- object avoidance

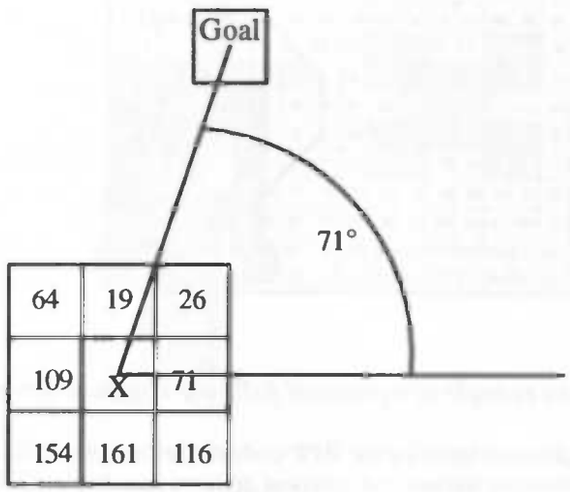



**Figure 7.6**: *The determination of the values added to the potential field. These values are determined by calculating the angle between the point under inspection and the goal point.*

A drawback of the potential field calculated as described above, is that it suffers from local minima. The calculated path consists of all the visited pixels from the initial start to the goal position. So at this moment each pixel of the path is an anchor. The routines of Hans Seinhorst are able to plan natural motions between anchors, but if the route has to be of a natural shape, only the crucial anchors have to be used. And of course, the less anchors, the faster the calculation of the natural path. This means that all redundant information has to be removed. The first filtering step is done during following the path of steepest descent. All pixels that belong to a straight line segment are deleted. The start– and endpoint of such a segment become anchors. In other words, while following the path of steepest descent, an anchor is placed when the path direction is changed (see Figure 7.7).

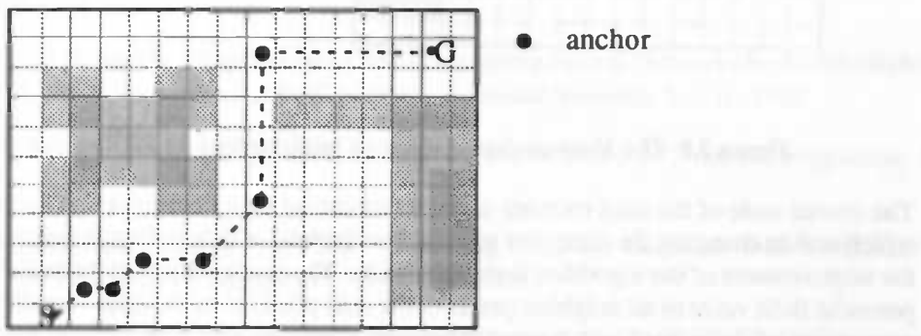



**Figure 7.7**: *Placing anchors at the begin– and endpoint of straight line segments*

By using these anchors the results looked quite well. But a situation as can be seen in Figure 7.8 can occur. A lot of anchors are placed, but only two anchors are needed. When using all the anchors (as in Figure 7.8) the walk of the ego object will look very silly. Each step will be succeeded by a right or left turn of 45 degrees, while the client expect a straight walk to the goal. This effect, the staircase effect, occurs when drawing lines in a grid representation. When the direction of the line doesn't fit with one of the directions from the starting pixel to the neighbor pixel this effect will occur. However, we like to represent a straight

line, in every direction, with one anchor for the begin point of the straight line and one anchor for the end-point.

We solved this problem by drawing a line from the first anchor to the third anchor. If the second anchor lies on this line within a certain threshold, this second anchor will be deleted. Then a line will be drawn between the first anchor and the new third anchor (the old third anchor becomes the new second anchor) and the process starts again. If the second anchor doesn't lie on the line between the first and third anchor, the first anchor will be saved. The second anchor becomes the new first anchor and the process will start again till the last anchor is processed. In Figure 7.8 this will result in two anchors, the start and goal position. By using this method it's possible that by clearing some anchors, corners of objects (or worse, whole objects) are crossed. Because this will not happen that often and there are no serious consequences (it just looks silly), we take this risk for granted. The overall results of this added anchor–filtering algorithm are very satisfactory.

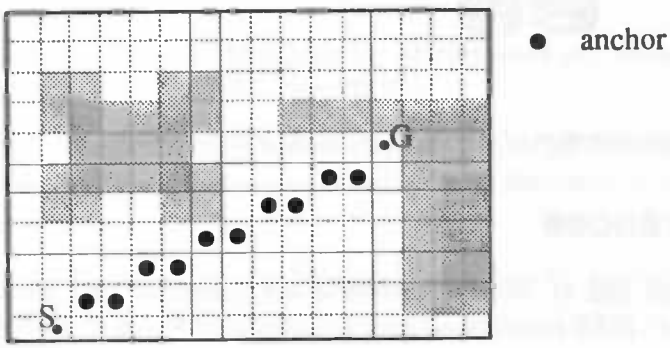



**Figure 7.8**: *The staircase effect causes too many anchors*

At this moment all the anchors are available and so the motions can be planned. As said before the motions will be planned by using the routines developed by Hans Seinhorst. These planned motion are stored in a linked list of atomic motions. The atomic motions are executed one by one by using a timer. We decided to use a timer to be able to handle other processes as well. In this way the rendering of the scene (also using a timer) will automatically be done. In this way it is also easy to implement the moving of several objects at the same time. To implement several moving objects, we use a linked list of all moving objects. An entry in this moving–objects list contains the linked list of atomic motions for this object. When a motion–set is planned it will be concatenated at the end of the linked motion list of an object. At every signal of the timer, the first atomic motions of all the moving objects will be executed.

## 7.5 K3 Motion in the future

At this moment, objects move in the virtual world from a start to a goal position by following a natural shaped path. In the future the limbs of the objects also have to move in a natural way. This could cause modifications to the object administration. It is very likely that for moving the limbs of the objects, extra information about the sub–objects, representing the limbs, is needed.

The 3D human being object, as it is currently used, can be seen in Figure 7.9. This object is not very realistic, but useful. We have searched for better 3D humanoid models, but couldn't find a useful one. There are a lot of 3D humanoid models available, but they aren't made of separate sub–objects. By using this kind of 3D models, we still can't move the limbs. A realistic model, in which all the limbs are represented by sub–objects, is probably not freely available.

In the future the path planning method also has to be optimized or replaced. The current method suffers from the local minima problem. In a commercial application, this is not acceptable. This chapter probably offers enough ideas for improvements or even the development of another path planning method.

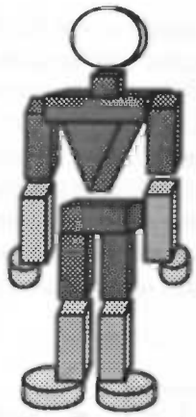


**Figure 7.9**: *The 3D representation of a human being, as it is currently used in the K3 User Program*

## Chapter References


[1] L. Gonzalez Sotelino and H. Bersini, "Hopfield net generation and encoding of trajectories in constrained environment", *IEEE International Joint Conference on Neural Networks*, Vol. II, pp. 857 – 862, 1992.

[2] T. Lozano–Perez, "Automatic Planning Manipulator Transfer Movements", *Robot Motion: Planning and Control, ed. Brady et al. Cambridge, Mass.: MIT Press*, 1982.

[3] R. Sutton, "Integrated Architectures for Learning and Reacting Based on Approximating Dynamic Programming", *Proceedings of the SAB conference in Paris: MIT Press*, 1990.

[4] M. Lemmon, "2–Degree–of–freedom Robot Path Planning using Cooperative Neural Fields", *Neural Computation,* 3, pp. 350–362, 1991.

[5] S. Najand, Z. Lo and B. Bavarian, "Application of Self–Organizing Neural Networks for Mobile Robot Environment Learning", *IEEE International Joint Conference on Neural Networks*, Vol. II, 1992.

[6] B. Pearlmutter, "Learning state space trajectories in recurrent neural networks", *Neural Computation*, 1, pp. 263 – 269, 1989.

[7] J. Heikkonen, P. Koikkalainen and E. Oja, "Self–Organizing Maps for Collision–free Navigation", *World Congres On Neural Networks*, Vol. III, pp. 141 – 144, July 1993.

[8] B. Baginski, M. Eldracher, "Path Planning with Neural Subgoal Search", *IEEE World Congres on Computational Intelligence*, Vol. V, pp. 2732 – 2736, 1994.

[9] J.F. Gilmore and J. Czuchry, "A Neural Network Model For Route Planning Constraint Integration", *IEEE International Joint Conference on Neural Networks*, Vol. III, pp. 221 – 226, 1992.

[10] A.A. Kassim and B.V.K. Vijaya Kumar, "A Neural Network Architecture for Path Planning", *IEEE International Joint Conference on Neural Networks*, Vol. II, pp. 787 – 792, 1992.

[11] S. Quinlan and O. Khatib, "", *Experimental Robotics II: Lecture Notes in Control and Information Sciences 190*, pp. 241–254, Springer–Verlag

[12] A. Zelinsky and S. Yuta, "A Unified Approach to Planning, Sensing and Navigation for Mobile Robots", *Experimental Robotics III: Lecture Notes in Control and Information Sciences 200*, pp. 444 – 455, Springer–Verlag

[13] H. Bersini and L.G. Sotelino, "A Simple Connectionist Model of Actions Sequence and Active Vision", *World Congres On Neural Networks*, Vol. III, pp. 212 – 215, July 1993.

[14] A.G. Barto, R.S. Sutton and C.J.C.H. Watkins, "Sequential Decisions Problems and Neural Networks", *Advances in Neural Information Processing Systems 2*, Ed. D.D. Touretzky, 1990

[15] R.S. Sutton, "Reinforcement Learning Architectures for Animats", *Proceedings of the First SAB Conference*, MIT Press 1990.

[16] H. Bersini, "Immune Network and Adaptive Control", *Proceedings of the first European Conference on Artificial Life*, MIT Press 1992.

[17] O. Khatib, "Real–Time Obstacle Avoidance for Manipulators and Mobile Robots", *International Journal of Robotics Research*, Vol. 5 No. 1, pp. 90 – 98, 1986.

[18] J. Boreinstein and Y. Koren, "Real–Time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments", *Proceedings of IEEE International Conference on Robotics and Automation*, Cincinatti, pp. 572 – 577, May 1990.

[19] R. Chattergy, "Some Heuristics for the Navigation of a Robot", *International Journal of Robotics Research*, Vol. 4 No. 1, pp. 59 – 66, 1985.

[20] A. Zelinsky, "Environment Exploration and Path Planning Algorithms for a Mobile Robot using Sonar", Phd dissertation, University of Wollongong, Department of Computing Science, October 1991.

[21] J. Barraquand and J.C. Latombe, "Robot Motion Planning: A Distributed Representation Approach", *International Journal of Robotics Research*, Vol. 10 No. 6, pp. 628 –649, 1991.

[22] A.G. Barto, R.S. Sutton and C.W. Anderson, "Neuronlike Adaptive Elements that Can Solve Difficult Control Problems", *IEEE Trans. on Systems, Man and Cybernetics*, Vol. 13, pp. 834 – 846, 1983.

[23] B.J.A. Kröse and J.W.M. van Dam, "Adaptive State Space Quantisation: Adding and Removing Neurons", *Artificial Neural Networks 2*, Ed. I. Aleksander and J, Taylor, Vol. I, pp. 619 – 622, 1992.

[24] J. del R. Millán and C. Torras, "Learning to Avoid Obstacles through Reinforcement", *Proceedings of the 8th International Workshop on Machine Learning*, pp. 298 – 302, 1991.

[25] J. Park and S. Lee, "Neural Computation for Collision–free Path Planning", *Proceedings of the International Joint Conference on Neural Networks*, Vol. II, pp. 229 – 232, 1990.

[26] H.A. Grave and J.H.M. Seinhorst, "Fuzzy–Based Motion Editing and Composing", *Master's Thesis, University of Groningen, August 1995.*

# Chapter 8 Client analysis

## 8.1 Introduction

When a salesman is trying to sell a product to a client, he must have some information about the client for his sales strategy. It is useless to try to sell a $20.000 kitchen appliance to a poor man who is only interested in cheap appliances. The salesman will have an initial impression of the client and can gather additional information by asking questions. With this information, the salesman builds a mental model of the client which he uses to determine which sales strategy to use.

The K3 User Program doesn't have a first impression of the client, but it can ask questions. So the questions to the client are more elaborate than the questions of the salesman to compensate for not being able to see the client. But questions aren't the only way to gather information about a client. Another good way of getting information from a client is to analyze its behavior in the virtual world. If a client is choosing expensive products, it will probably mean that the client has an expensive taste. From this information, the sales strategy can be altered to try to sell more expensive products.

From this information a model of the client is build. This model consists of abstract keywords that describe a client and are called the *client characteristics*. These client characteristics are used to decide which sales strategy to use.

To gather the client information, we have created the Intelligent Person Identification (IPI) and the Intelligent Behavior Identification (IBI). The IPI can ask questions to the client and from the answers it determines the client characteristics, which are used to determine a sales strategy. The IBI analyses the actions of the client in the virtual world. This analysis can be used to adjust the sales strategy. The answers from the IPI and the additional information from the IBI are stored in the so–called *client profile*. By doing this, we don't have to ask a client the same question twice.

In this chapter, we will describe how the IPI and the IBI work. Since a major part of the IPI consists of neural networks, we will start with a brief introduction to neural networks.

## 8.2 Neural networks

In this section we will describe what neural networks are. We will give the benefits of neural networks and give a model of a neural network to introduce the terminology we use to describe neural networks. This brief introduction to neural networks comes from [1] where this and more about neural networks is discussed in full detail.

### 8.2.1 What is a neural network

Work on neural networks has been motivated by the recognition that the brain computes in an entirely different way from the conventional digital computer. The struggle to understand the brain owes much to the pioneering work of Ramón y Cajál ([2]), who introduced the idea of *neurons* as structural constituents of the brain. The brain is a highly *complex, nonlinear, and parallel computer* (information–processing system). It has the capability of organizing neurons so as to perform certain computations (e.g. pattern recognition, perception, and motor control) many times faster than the fastest digital computer in existence today. How does a human brain do it? At birth, a brain has great structure and the ability to build up its own rules through what we usually refer to as"experience." Indeed, experience is built up over the years. A developing neuron is synonymous with a plastic brain: *Plasticity* permits the developing nervous system to adapt to its surrounding environment ([3]; [4]). In an adult brain, plasticity may be accounted for by two mechanisms: the creation of new connection between neurons, and the modification of existing connections. Just as plasticity appears to be essential to the functioning of neurons as information–processing units in the human brain, so it is with neural networks made up of artificial neurons. In its most general form, a *neural network* is a machine that is designed to *model* the way in which the brain performs a particular task or function of interest. To achieve good performance, neural networks employ a massive interconnection of simple computing cells referred to as "neurons". We may thus offer the following definition of a neural network viewed as an adaptive machine ([5]):

*A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:*

1. *Knowledge is acquired by the network through a learning process.*
2. *Interneuron connection strengths known as synaptic weights are used to store the knowledge.*

A neural network derives its computing power through its massively parallel distributed structure and its ability to learn and therefore generalize. Generalization refers to the neural network producing reasonable outputs for inputs not encountered during training (learning). These two information–processing capabilities make it possible for neural networks to solve complex problems.

### 8.2.2 Benefits of Neural Networks

The use of neural networks offers us the following useful properties and capabilities:

1. *Nonlinearity.* Since a neuron is basically a nonlinear device, a neural network is itself nonlinear. This is an important property, particularly if the underlying physical mechanism responsible for the generation of an input signal is inherently nonlinear.
2. *Input–Output Mapping.* For our neural networks we use the popular paradigm of learning called *supervised learning*. This involves the modification of the synaptic weights of a neural network by applying a set of examples, each consisting of a unique *input signal* and the corresponding *desired response*. The synaptic weights are modified so as to minimize the difference between the desired response and the actual response of the network produced by the input signal. Thus the neural network learns from the examples by constructing an *input–output mapping* for the problem at hand.
3. *Adaptivity.* Neural networks have a built–in capability to *adapt* their synaptic weights to changes in the surrounding environment. In particular, a neural network trained to operate in a specific environment can be easily *retrained* to deal with minor changes in the operating environmental conditions.
4. *Evidential Response.* In the context of pattern classification, a neural network can be designed to provide information not only about which particular pattern to *select*, but also about the *confidence* in the decision made.
5. *Contextual Information.* Knowledge is represented by the very structure and activation state of a neural network. Every neuron in the network is potentially affected by the global activity of all

other neurons in the network. Consequently, contextual information is dealt with naturally by a neural network.

### 8.2.3 Model of a Neuron

A *neuron* is an information–processing unit that is fundamental to the operation of a neural network. Figure 8.1 shows the *model* for a neuron. We may identify three basic elements of the neural model:

1. A set of *synapses* or *connection links*, each of which is characterized by a *weight* of its own. Each input is multiplied by its weight.
2. An *adder* for summing the input signals multiplied by their weights.
3. An *activation function* for limiting the amplitude of the output of a neuron.

In mathematical terms, we may describe a neuron k by writing the following pair of equations:

$$u_k = \sum_{j=1}^{p} w_{kj} x_j \tag{1.1}$$

$$y_k = \phi(u_k - \theta_k) \tag{1.2}$$

where $x_1, x_2, ..., x_p$ are the input signals; $w_{k1}, w_{k2}, ..., w_{kp}$ are the synaptic weights of neuron $k$; $u_k$ is the *linear combiner output*; $\theta_k$ is the *threshold*; $\phi(.)$ is the *activation function*; and $y_k$ is the output signal of the neuron.

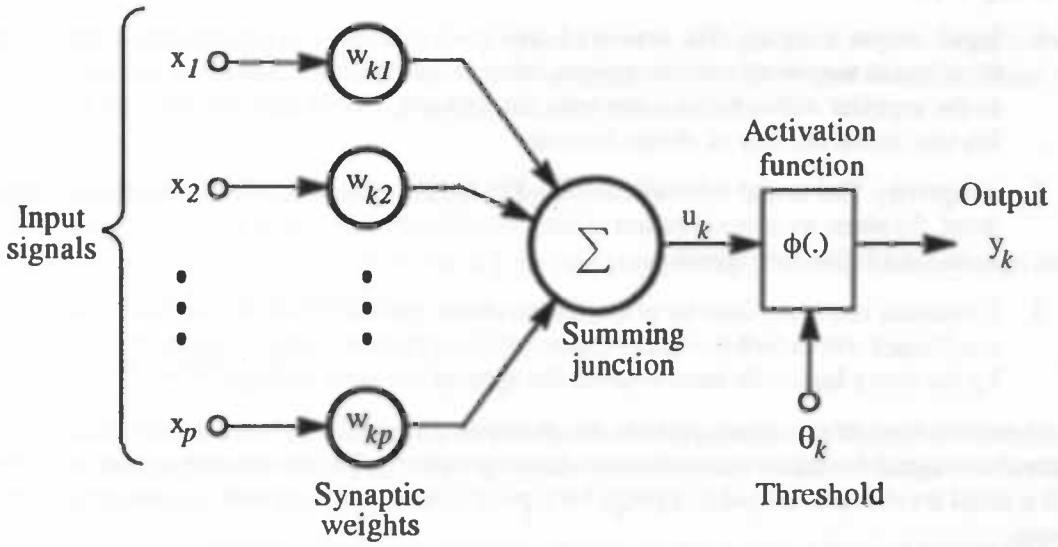



**Figure 8.1:** *Nonlinear model of a neuron.*

## 8.3 Intelligent Person Identifier (IPI)

The IPI has to give a sales strategy to the Sales Engine. To be able to do this, the IPI has to know something about the client. The IPI has two ways of getting this information:

1. examine the client profile,
2. ask the client.

Since we want to get as much information about the client without having to ask him, our first source of information is the client profile. The K3 User Program has stored all facts about the client from previous

sessions in the client profile. In this way, the client won't be asked the same questions over and over again. Even if the client has never contacted this supplier, the information is still available.

All the facts the IPI has to have from the client in order to determine a sales strategy, are summed up in a questionnaire. This questionnaire is filled in with the facts from the client profile. This can leave some questions unanswered. All unanswered questions are then asked to the client and the answers are stored in the client profile.

There are two ways in which we can ask a question:

1. direct: e.g. pop–up a question form and let the client type in his answer, or
2. indirect: let the client respond to a given situation and let his reaction be the answer.

The first way is trivial, but the second way implies we have to create a situation and analyze the response of the client until we have an answer. The indirect way provides us with a means to get answers to questions clients are reluctant to answer when they are asked directly.

Now the IPI has all the client information it needs from the client to determine a sales strategy. The next step is to build a model of the client. The model consists of abstract keywords, called *client characteristics*, describing the client. These client characteristics describe the client in such a way that a sales strategy can be determined.

To extract the client characteristics from the client profile, we use neural networks. As seen in paragraph 8.2.2, a neural network provides us with several useful properties and capabilities which will help us in the following way:

1. Input–output mapping. The network learns from examples. These examples can be provided by the client as they work with the system. After using the system, the client information is sent back to the supplier with which he can train the network. In this way, the network becomes better at his task as the number of clients increase.
2. Adaptivity. The neural network can be easily retrained to deal with minor changes in the environment. So when we add a question to the questionnaire, the neural network can be easily retrained to deal with this new question.
3. Evidential response. Besides giving us the client characteristics, the network can also give us the confidence with which this client characteristic is chosen. These confidence numbers can be used by the fuzzy logic rule base to select the appropriate sales strategy.

Another advantage of neural networks is the ability to generalize. This means that a network produces reasonable outputs for inputs not encountered during training. This means that we can train the network with a small set of input–output mappings for typical clients, without limiting ourselves to these typical clients.

For each client characteristic, we built a neural network. The sales manager, who has chosen the client characteristics, selects for each client characteristic the questions from the questionnaire which influence this client characteristic. The answers to these questions are used as inputs for the network for each client characteristic and are also stored in the client profile.

In [6], Toia describes his research into describing clients using client characteristics. These client characteristics describe the clients in a general way. Each sector has to add the client characteristics which are important for their sector. He has built several neural networks which extract the client characteristics from the client profile. Since neural networks expect numeric input, each input element from the client profile has to be mapped on a number between 0 and 1. For this, a fuzzy logic rule base can be used. So each client profile element which is used as input for a neural network has to be mapped into a number between 0 and 1. This has to be kept in mind when creating the questions for the IPI. Questions have to be asked in such a way that a finite number of discrete answers are possible.

The last step is to go from client characteristics to sales strategy. The sales manager has chosen the client characteristics, keeping in mind that a sales strategy has to be chosen from these client characteristics. So he must now be able to construct rules for a rule base which use the client characteristics as input and generate a sales strategy as output.

The IPI just describes is depicted in Figure 8.2. We can see that each client characteristic has its own neural network. Each neural network has its own set of inputs from the client profile. The client characteristics are used by the rule base to come up with a sales strategy.

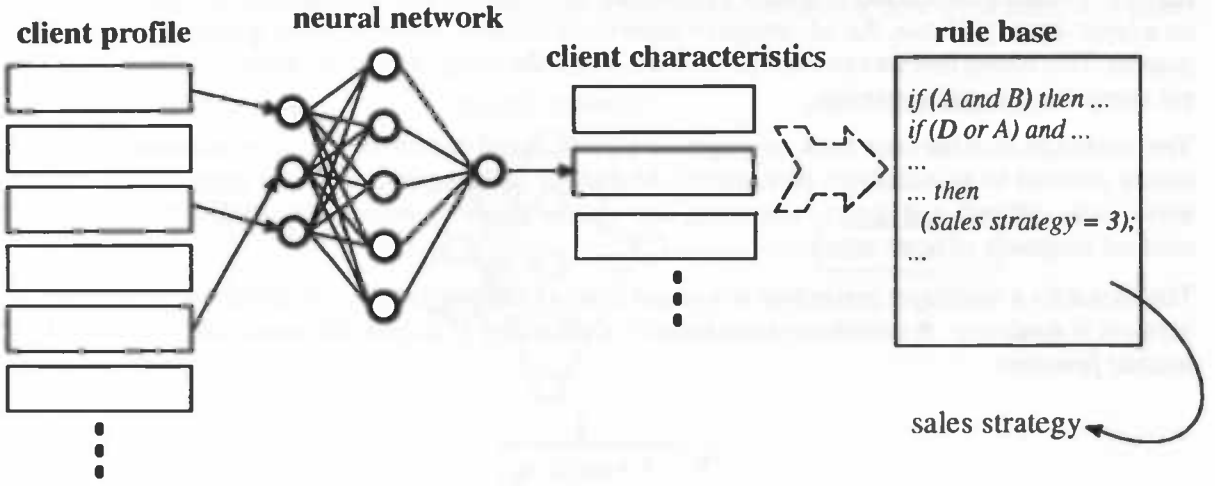



**Figure 8.2:** *Internal working of the IPI: client profile is used to provide input for neural networks, each neural network has a client characteristic as output, a rulebase determines the appropriate sales strategy by evaluating the client characteristics.*

## 8.4 Implementation aspects of IPI

In this section we will describe the parts of the IPI that are implemented. We will look at which kind of neural network we use and how to train it.

### 8.4.1 Neural networks

The manner in which the neurons of a neural network are structured is intimately linked with the learning algorithm used to train the network. In [1], Simon Haykin identifies four different classes of network architectures, namely:

1. Single–layer feedforward networks. This type of network has an *input layer* of source nodes that project on an *output layer* of neurons, but not vice versa.
2. Multilayer feedforward networks. This type of network has one or more *hidden layers*, which enables the network to extract higher–order statistics.
3. Recurrent networks. This type of neural network has at least one *feedback* loop, that is, the output of a neuron is the input of a neuron in the same or previous layers.
4. Lattice networks. This type of network is really a feedforward network with the output neurons arranged in rows and columns.

The choice for the network architecture depends on the learning algorithm used to train the network. In the next paragraph we will choose the learning algorithm and therefore choose the network architecture.

The neural network tool we have created provides us with a set of *input–output* examples provided by the sales manager. When we can provide the neural network with a *target* response for every input vector in the learning phase, we apply a so–called *supervised learning* technique. Here the network is exposed to an input vector and produces an output vector. From the target response and the output from the neural network, we can calculate an *error signal*. The weights in the neural network are adjusted with the aim of eventually making the neural network emulate the teacher. The learning algorithm used for making the adjustments to the weights is called the back–propagation algorithm. The back–propagation algorithm derives its name from the fact that error terms in the algorithm are back–propagated through the network, on a layer–by–layer basis. An advantage of supervised learning is that it can be performed in an *off–line* manner. This means that we can train the network, *freeze* the design and let the network operate on a different computer in a *static* manner.

The combination of the error back–propagation algorithm and multilayer feedforward networks are commonly referred to as *multilayer perceptrons*. Multilayer perceptrons have been applied successfully to solve some difficult and diverse problems, like optical character recognition, system identification or medical diagnosis of heart attacks.

The choice for a multilayer perceptron as a neural network implies that the activation function of the neural network is *nonlinear*. A commonly used form of nonlinearity is a *sigmoidal nonlinearity* defined by the *logistic function*:

$$y_k = \frac{1}{1 + \exp(-u_k)} \tag{1.3}$$

where $u_k$ is the net internal activity level of neuron $k$ and $y_k$ is the output of the neuron (see also Figure 8.1).

## 8.4.2 Neural networks in the K3 User Program

Once the sales manager has built a neural network (see following section), this network is *frozen*, that is the neural weights are recorded in a '.net'–file. This file contains the following keywords (the '*' mean one or more times):

- GROUPS: number_of_layers number_of_neurons*
- CONNECTIONS [(x, y) : float]*

The label GROUPS is followed by an integer number denoting the number of layers this neural network consist of, including the input layer and the output layer. So the integer *number_of_layers* denotes the summation of all layers of the neural network. The following numbers represent the number of neurons in each layer. The first number gives us the number of neurons in the first layer, the second number gives us the number of neurons in the second layer, and so on. This gives us a specification of how the neural network should be built. Each neuron is given an unique id: an integer number counting from one upwards and starting at the first input neuron.

The label CONNECTIONS is followed by a series of neuron pairs and a floating point number between zero and one. This floating point number is the synaptic weight between neuron $x$ and neuron $y$. Each pair of neurons that are connected are listed after the label CONNECTIONS with their respective synaptic weight. Now, for each connection between neurons, their respective synaptic weights can be filled in.

To evaluate an input vector, for each neuron $y$ we calculate the total sum of the products of each neuron in the previous layer and the synaptic weight factor between that neuron and neuron $y$. That is, we apply equation (1.1). Next, we apply the activation function as defined in equation (1.3). We do this for all neurons, starting with the neurons in first layer and ending with the neurons in the last layer. The values calculated for the neurons in the last layer give us the output vector.

## 8.4.3 Neural network tool for sales managers

One reason to analyze the client profile with neural networks is the adaptivity of a neural network. This means that a sales manager can alter the questionnaire, alter the client characteristics or add sales strategies and easily retrain the neural network to deal with the new information. For this purpose a neural network tool has been created. This tool is not a part of the K3 User Program, but part of the system that is available to the supplier. It was built to demonstrate the ease with which a neural network can be built and retrained.

To add a question (i.e. add a client profile element and therefore add an input to the neural networks), the sales manager gives the client characteristics which are influenced by this question. The neural networks for these client characteristics have to be retrained. This doesn't mean that the old data set, used to train the old network, has become useless. This data set can be updated to include the new input and can be used to train the new network.

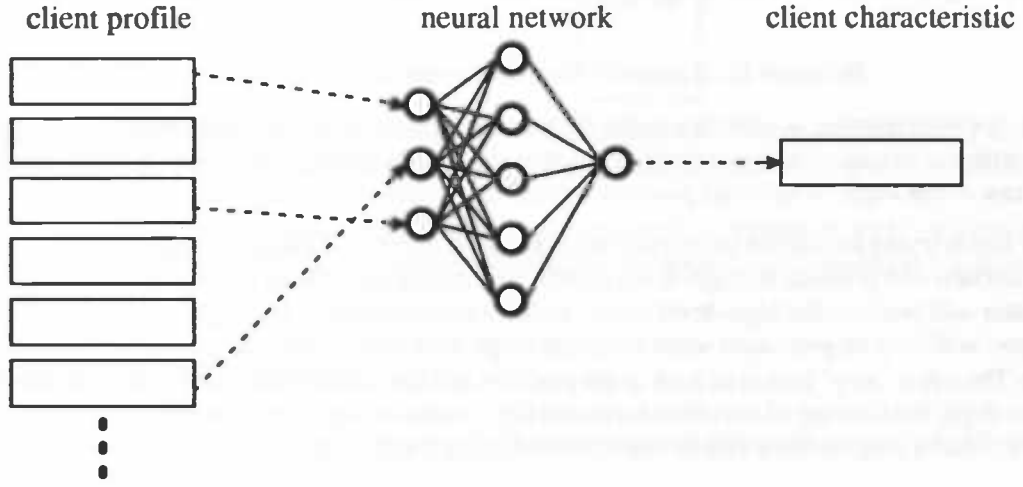



**Figure 8.3**: *Neural network for a client characteristic.*

Each client characteristic will have its own neural network. The inputs of the neural network will be the answers from the client, while the output will be the client characteristic. This is depicted in Figure 8.3. To create a client characteristic, just let the tool know which inputs from the client profile have an influence on the client characteristic, fill in some default values for typical clients and the tool will create and learn the neural network accordingly.

To add a new client characteristic, a neural network has to be made. The sales manager decides from which client profile elements this client characteristic can be deduced. He then has to fill in some default values for typical clients and the tool will create and learn the neural network.

A sales manager might discover, after examining the client information which is sent back to the supplier, that other types of clients than he expected are using the system. The neural networks have to be retrained to incorporate these clients. For each neural network, the sales manager has to fill in some typical values for these clients, after which the neural networks are retrained, using the old data set and the new information.

In this way we have created a tool for a sales manager with which he can create an intelligent tool without having to know anything about neural networks or having to maintain a large rule base. The sales manager just describes the typical customers in terms of their respective client characteristics.

Another reason to use neural networks is the learning aspect of neural networks. While clients are using the system, more and more information about the clients is sent back to the supplier. This information can be used to retrain the neural networks. The neural networks will become better and better at setting the client characteristics for each client. Also, the sales manager can start with the default sector networks and retrain the networks for his specific type of clients. And since the neural weights are sent to the client via a network, the clients will always work with the latest neural networks available.

## 8.5 Intelligent Behavior Identifier (IBI)

The task of the IBI is to analyze the behavior of the client to provide the Sales Engine with information. This information might even lead to changes in the client profile and thus to changes in the sales strategy.

Suppose, for example, a person is analyzed by the IPI to be a person who would buy a cheap kitchen. This person is now choosing expensive kitchen appliances. This might lead to changes in the client profile, like 'income'. This, in turn, might lead to changes in the sales strategy, since we are now dealing with someone who has more money to spend.

How can we analyze the behavior of the client? First we have to give a definition of behavior in the virtual world:

*Behavior is all input by the client in the virtual world.*

Not all behavior is interesting. A step forward in the virtual world or choosing a more expensive refrigerator are two different events, the former being uninteresting, the latter being interesting. So the first step is to filter all raw client input to leave all possibly interesting events to be analyzed.

The second step is trying to convert the events into high–level event. For example, stopping in the middle of a virtual kitchen will produce no high–level events, while stopping in front of a refrigerator and facing the refrigerator will produce the high–level event 'looking at refrigerator.' The high–level event 'looking at refrigerator' will only be generated when the client stops, is in front of the refrigerator and is facing the refrigerator. The event 'stop' makes us look at the position and direction of the client in the virtual world. The first two steps, the filtering of the client input and the creation of high–level events, is done by the IBI preprocessor. Such a preprocessor can be implemented using a rule base.

It is now possible to create another rule base which acts upon these high–level events. Suppose a client is watching a microwave for more than 30 seconds. It might now be wise to give a demonstration of the microwave. Or suppose a client has chosen an expensive microwave, a dish washer, but also a cheap refrigerator. Then is might be a good idea to demonstrate a more expensive refrigerator.

These rules can be put into a fuzzy logic rule base. So the third step is to use the high–level events as input for a fuzzy logic rule base. The output of the fuzzy logic rule base might look like 'demonstrate microwave, 0.80'. The output 'demonstrate microwave' is given a confidence value. Only if the confidence value is high enough the output is taken into account. The last step is to take the output with the highest confidence value and send it to the Sales Engine. The Sales Engine can use this information to change the scenario stack (see also chapter 5).

As we have seen, the behavior of the client might also lead to changes in the client profile. When this occurs, we have to activate the IPI again to see if the sales strategy has changed. Figure 8.4 depicts the IBI just described.

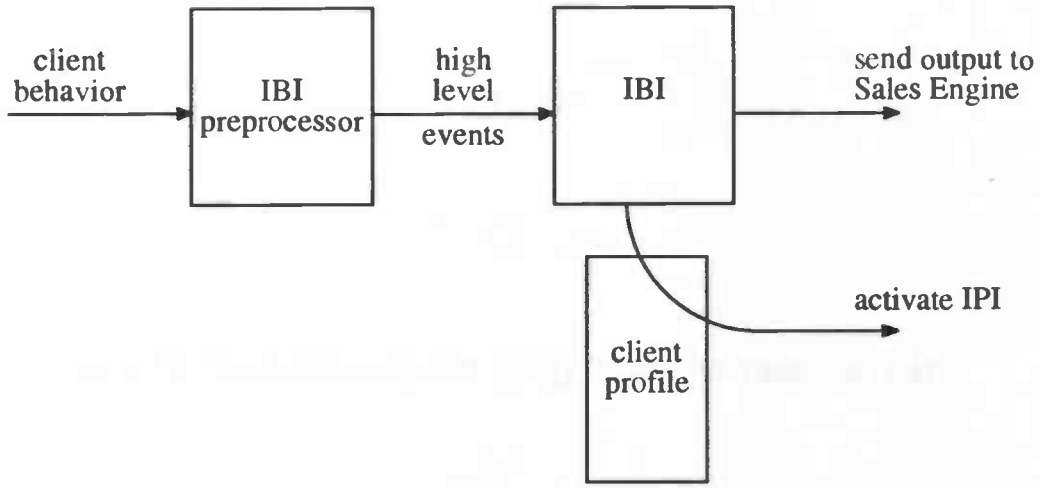



**Figure 8.4**: *The client behavior is filtered to create the high–level events for the IBI. If the IBI makes changes to the client profile it activates the IPI, otherwise it sends its output to the Sales Engine.*

Since we use terms as 'refrigerator' and 'microwave' in stead of 'Whirlpool 3000', we can use a general rule base which can be used by each supplier in the sector. As always, each supplier is free to create his own rule base.

## Chapter references

[1] Simon Haykin, 1994. "Neural Networks, A Comprehensive Foundation", *New York: Macmillan College Publishing Company.*

[2] Ramón y Cajál, S., 1911. "Histologie du système nerveux de l'homme et des vertébrés", *Paris: Maloine; Edition Francaise Revue: Tome I, 1952; Tome II, 1955; Madrid: Consejo Superior de Investigaciones Cientificas.*

[3] Churchland, P.S., and T.J. Sejnowski, 1992. "The Computational Brain", *MA: MIT Press.*

[4] Eggermont, J.J., 1990. "The Correlative Brain: Theory and Experiment in Neural Interaction", *New York: Springer–Verlag.*

[5] Aleksander, I., and H. Morton, 1990. "An Introduction to Neural Computing", *London: Chapman & Hall.*

[6] L.F. Toia, "ITS MAGIC, Laboratorium 9 – Verslag", *University of Groningen, April 1996.*

[7] B. Kuipers, "Neurale Netwerken in expertsystemen", *Master's Thesis, University of Groningen, May 1995.*

# Chapter 9 Conclusions and future research

The main goal of the research was to find out if an interactive tele–shopping application, as described in chapter 1, could be implemented. By developing a prototype that meets the requirements set forth in the aforementioned chapter, we have shown that such an application can indeed be implemented.

When we compare a conventional multi–media presentation to a presentation using the ITS MAGIC architecture, we see that the amount of data needed for the latter is relatively small. Given the small network bandwidth available to Internet users and the lasting increase in both the number of systems and the number of users on the Internet, possibly leading to a decrease in available bandwidth per user, this should be considered an important benefit.

During the specification phase, SADT has proven to be a very useful formal specification method. With its ability to depict a part of the system on one page, in a clear and natural way, it has helped us to describe and discuss our ideas about the system down to the desired level of detail. As a result of this, the implementation phase became relatively straightforward, which, in our view, describes the importance of a proper specification phase.

As expected, the use of ADTs has proven to be a good method for structurizing the implementation. However, it does require a large degree of discipline to all workers on the project, because it does not enforce the use of access routines for data types instead of accessing the data types directly. For this, Object Orientation tools may provide a better method. In our case, ADTs were used very strictly, which made a parallel development of the application possible.

We have implemented part of the intelligent client analysis. The results so far provide enough reasons to justify future research and development in this area. We also started working on an intelligent tool for Sales Managers, which automatically creates an intelligent personality identificator for a given situation. For this, the Sales Manager only has to provide some examples for different client types. We think that the neural network approach used in this tool can be very useful, but more research in this area is needed.

Lastly, we hope that the parts of the ITS MAGIC system that are currently not, or only partially, implemented can be implemented, so that the full strength of the ITS MAGIC approach can be used for real–life applications.