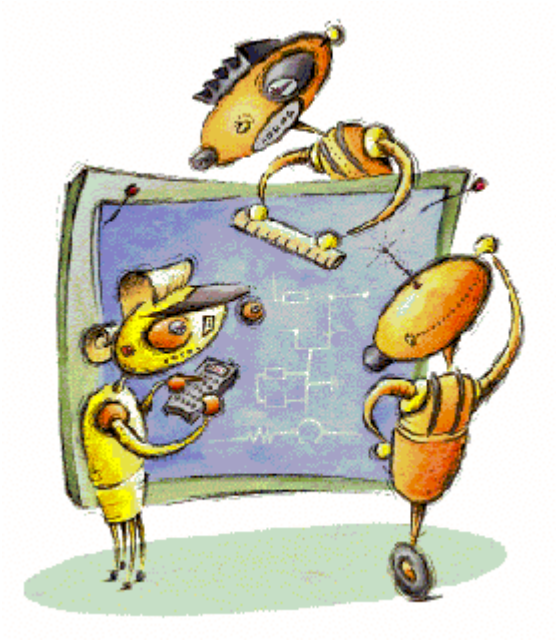
Intelligente agent-systemen

Dé innovatie in automatisering



BWI Werkstuk

juni 2011

Robert Post

Begeleider: prof. dr. R. van der Mei

Vrije Universiteit
Faculteit der Exacte Wetenschappen
Bedrijfswiskunde en Informatica
De Boelelaan 1081a
1081 HV Amsterdam

vrije Universiteit

Inhoudsopgave

Inleiding

Auto's die voor ons inparkeren, verkeersborden lezen en afstand bewaren. Het Amerikaanse leger dat autonome onbemande voertuigen wil inzetten[1]. Complexe, dynamische bedrijfsprocessen die autonoom draaien. Een technologie waarop deze systemen kunnen draaien zijn *intelligente agent-systemen*: systemen die in staat zijn autonoom te denken en te handelen. Het werkstuk vindt een antwoord op vragen als: 'Wat zijn intelligente agent-systemen?', 'Hoe zijn deze systemen ontstaan (wat is hun geschiedenis)?', 'Waarom is dit dé innovatie in automatisering?', 'Hoe worden deze systemen ontwikkeld (gemaakt)?' en 'Wat is het grote voordeel van de samenwerking van deze systemen?'.

Door steeds complexer wordende (bedrijfs)processen is steeds meer automatisering en autonomie van deze processen vereist. Agenten komen tegemoet aan de huidige behoeften en aan de toekomstige behoeften: de toepassingen van agenten groeien mee met de technologische ontwikkelingen in de maatschappij.

Het idee van intelligente agent-systemen, ook wel agent-systemen of kortweg agenten genoemd, is ontstaan in de Kunstmatige Intelligentie. Daarnaast heeft het veel raakvlakken met andere gebieden omdat zaken zoals modelleren, wiskundige vraagstukken, complexe systemen en samenlevende componenten aan bod komen. Intelligentie in deze systemen is een belangrijk begrip, daarom is Deel I van het werkstuk gericht op het ontstaan van intelligentie in systemen. Deze geschiedenis gaat terug tot de tijd van de oude Grieken, die zich al bezig hielden met het idee van intelligentie. Hierna wordt er ingegaan op het ontstaan van de intelligente agent-systemen.

In Deel II wordt ingegaan op wat agenten zijn. Er blijkt in de literatuur geen eenduidigheid te zijn over de precieze betekenis van agenten, juist vanwege de vele toepassingen in de verschillende vakgebieden die gebruik maken van de agent-systemen. Het werkstuk zal in Deel II een algemeen beeld geven van agent-systemen: Wat zijn deze systemen? Waarom en hoe worden ze gemaakt? Waar worden ze voor gebruikt?

Tot slot komen in Deel III multi-agent-systemen aan bod. De meest complexe situaties kunnen aangepakt worden met deze multi-agent-systemen, waarin een groot probleem verspreid wordt over meerdere agenten en dus in meerdere subproblemen. Om dit goed te laten verlopen, vereist een multi-agent-systeem interactie tussen de agenten. Belangrijk onderdeel van de interactie is coördinatie. De coördinatie die plaatsvindt tussen agenten houdt niet alleen samenwerking voor een gezamenlijk doel in, maar ook het wedijveren om bepaalde resources. Verder worden er twee ideeën besproken die de coördinatie tussen de agenten kunnen bewerkstelligen.

[1] http://www.defencetalk.com/us-navy-marines-test-autonomous-unmanned-ground-vehicles-27996/ (unmanned vehicles US army)

Deel I – Geschiedenis van intelligentie (in systemen)

Om te begrijpen wat intelligente agent-systemen zijn, is het interessant om onderzoek te doen naar de ontstaansgeschiedenis van deze complexe systemen. In het eerste deel van het werkstuk zal aan bod komen hoe mensen over intelligentie dachten door de eeuwen heen, en hoe uiteindelijk het idee van agent-systemen ontstaan is.

Hoofdstuk 1 - De ontwikkeling op het gebied van programmeren

1.1 Introductie

Tegenwoordig wordt steeds meer (alledaagse) apparatuur met processoren en chips gebouwd. Deze stellen de apparatuur in staat om veel meer te kunnen dan voorheen en zelfs om zelf na te denken. Ze kunnen vandaag de dag op zo'n manier geprogrammeerd worden dat ze zelfstandig kunnen handelen. Daardoor kunnen er systemen (samenwerkende componenten) ontstaan op allerlei gebieden die draaien zonder menselijke interventie. Wat niet wil zeggen dat mensen geen onderdeel van zulke systemen kunnen zijn. De ontwikkeling op dit gebied heeft zich vooral de laatste jaren in snel tempo voltrokken.

1.2 Overzicht van de ontwikkeling van de wijze van programmeren

In Figuur 1, een samenvoeging van [37] en [4], is er een verschuiving te zien van machine georiënteerd programmeren naar een programmeerwijze die gebaseerd is op cognitief denken, de menselijk georiënteerde manier van problemen oplossen. De periodes geven aan wanneer het belangrijkste deel van het werk is gedaan.

… - 1960 Machinecode en Assembleertaal
1960 - 1970 Machine onafhankelijke programmeer talen Procedure gericht programmeren [*sub routines*]
1970 - 1980 Gestructureerd programmeren [*procedures & functies*]
1980 - 1990 Objectgeoriënteerd programmeren Declaratief programmeren [*met logica, wat te doen ipv. hoe* (bijv. *SQL)*]
1990 - 2000 Frameworks [*API*], scenario's, design patterns [*templates*], en protocollen [*voor communicatie*]
2000 – … Zelfdenkende systemen (KI-systemen) zoals: Agenten Multi-Agent-systemen

Figuur 1: overzicht van de ontwikkeling van de wijze van programmeren.

Hoofdstuk 2 - De geschiedenis van agenten[2]

2.1 Ontstaan van de wetenschap Kunstmatige Intelligentie

Zoals te zien in het overzicht zijn de nieuwste systemen, systemen die op zogenaamde intelligente agenten draaien. Het ontstaan en de evolutie van deze agenten is vooral toe te schrijven aan de ontwikkelingen op het gebied van Kunstmatige Intelligentie (KI). Deze wetenschapstak is zelf ook nog erg jong en is ontstaan in de jaren vijftig.

Al eeuwen voordat KI als wetenschapstak ontstond, was men bezig met het concept intelligentie. Aristoteles (5e eeuw) ontwikkelde het zogenaamde syllogisme; een formele vorm van redeneren waarin conclusies kunnen worden getrokken uit bepaalde gegeven vooronderstellingen (zgn. premissen)[3].

Het werk van Aristoteles op dit gebied maar ook op vele andere gebieden is eeuwenlang bestudeerd, doorontwikkeld en onderwezen. Onder andere door de 13e eeuwse Albert Magnus en Roger Bacon, waarover het verhaal gaat dat ze zogenaamde 'talking heads' bezaten, profetische intelligente metalen hoofden die kunnen horen, denken en begrijpen als mensen en zo elke vraag die ze gesteld wordt correct kunnen beantwoorden. Ondanks dat het verhaal van dit intelligente mechanisme onzin is, is hier wel een voorloper ontstaan van het idee over robots, cyborgs en kunstmatige intelligentie. Ook creëerde Ramon Llull in de 13e eeuw apparaten die niet-wiskundige waarheden toonden, op basis van (een voorloper van) combinatoriek. De kennis kwam voort uit gecombineerde logica, dit proces kan dus beschouwd worden als een soort van kunstmatige intelligentie.

Een verdere basis van de ontwikkeling op het gebied van KI was in de 15e en 16e eeuw de uitvinding van de klok, en dan vooral de techniek waarmee de klok 'loopt'. Met behulp van deze techniek konden ook mechanische dieren worden gemaakt oftewel de eerste robots.

De volgende ontwikkeling, een revolutie op het gebied van denken over het denken, kwam in de 17e eeuw beginnend met Descartes, die dieren voorstelde als complexe machines, hierop baseerde Turing onder andere zijn werk. Maar ook andere kwamen met vernieuwingen: Hobbes op het gebied van combinatorisch denken, Pascal vond de eerste mechanische digitale rekenmachine uit, waarna Leibniz deze machine verder uitbreidde. In de eeuw daarna werden steeds meer mechanische uitvindingen gedaan in de vorm van speelgoed (mechanische eend van Vaucanson en von Kempelen schaakspeel machine, ook wel de Turk genaamd). In de 19e eeuw kwam George Boole met de Boolean logica. Een vorm van deze logica, de twee-waarden Boolean Logica, waarin de onderdelen worden aangeduid met 0 en 1 (of false en true) bleek later erg handig te zijn in het beschrijven van de operaties van elektrische schakel circuits (computer chips en geïntegreerde circuits).

Uit bovenstaande ontwikkelingen komt duidelijk naar voren dat er door de eeuwen heen een fundament is gezocht om het begrip intelligentie te vangen en te beschrijven. Dit om machines te ontwikkelen die voor zichzelf kunnen denken, zoals mensen dat kunnen.
Toch was er pas in de jaren 50 voldoende basis om aan 'echte' kunstmatige intelligentie te denken. Het voorwerk hierin werd gedaan op het gebied van neurologie: er werd ontdekt dat het brein een

2 Hoofdstuk 2 is onder andere gebaseerd op [3], [5], [17],[26], [27] en wikipedia pagina's over de specifieke onderwerpen/personen.

3 Vb: Alle mensen zijn sterfelijk (majorpremisse)
Socrates is een mens (minorpremisse)
Socrates is sterfelijk (conclusie)

neuraal netwerk is. Door deze ontdekking te combineren met het werk op het gebied van digitalisering ontstond het idee van een elektronisch brein. In 1951 bouwde Minsky de SNARC, een machine, voor wiskundige berekeningen, gebaseerd op een neuraal netwerk.

Voor deze tijd waren de ENIAC (1945) de eerste elektronische digitale computer en de EDVAC (1951) de geavanceerdere opvolger, met onder andere werkgeheugen, ontwikkeld. Dit waren een van de eerste supercomputers, gebruikt voor het berekenen van complexe wiskundige systemen. Het was ook in deze tijd dat de vraag rees in de informatica wereld of machines kunnen denken en dus of machines intelligent zijn. Alan Turing (1912 – 1954) bedacht een test om aan te tonen of een machine inderdaad beschouwd kan worden als intelligent: de Turing Test (voor uitleg over de test zie Bijlage 1.1). Doordat er nu aan de ene kant de rekenkracht was en aan de andere kant het idee van intelligentie in machines, werd in 1956 de Darthmouth conferentie gehouden. Aanwezig waren onder andere McCarthy, Minsky, Newell en Simon. Het doel van deze bijeenkomst was “een studie starten voor het zo precies mogelijk beschrijven van alle aspecten van intelligentie, zodat deze geïmplementeerd kan worden in een machine”. Op deze conferentie toonden Newell, Shaw en Simon de eerste KI computer: de Logical Theorist [28]. Deze computer kan het menselijk handelen nabootsen van probleemoplossen door onder andere te werken met symbolen in plaats van getallen. Een symbool systeem (of formeel systeem) neemt fysieke patronen (de symbolen), combineert deze in structuren (expressies) en voert manipulaties uit door middel van processen op de symbolen om nieuwe expressies te produceren. The physical symbol system hypothesis (PSSH) geformuleerd door Newell en Simon zegt: “Een fysiek symbol systeem bezit de noodzakelijke en de voldoende voorwaarden (twee termen uit de logica) om te zorgen voor algemene intelligentie actie.” De hypothese houdt in dat het menselijk denken gebaseerd is op symbool manipulatie (vanwege de noodzakelijke voorwaarde is er voor intelligentie een symbool systeem nodig). Verder zegt de hypothese dat machines intelligent kunnen zijn omdat een symbool systeem een voldoende voorwaarde is voor intelligentie [29]. Voorbeelden van fysieke symbool systemen zijn te vinden in Bijlage 1.2 (Engels).

De Logical Theorist was uiteindelijk in staat om “38 van de 52 theorieën in Whitehead en Russell's *Principia Mathematica*[4] te bewijzen”. Deze conferentie, met al z’n nieuwe bevindingen en grote namen uit de wetenschap, wordt beschouwd als de geboorte van de KI.

Na deze conferentie braken de gouden jaren (1956 – 1974) in de KI aan. Door de grote potentie die men zag werd er ontzettend veel geld gestopt in KI centra die opgericht werden door o.a. McCarthy & Minsky en Newell & Simon, waardoor die centra vele resultaten opleverden.

Newell, Shaw en Simon bouwden een programma, de General Problem Solver (GPS), met daarin heuristieken om de search space van een probleem (alle mogelijke paden naar een mogelijke oplossing) te reduceren. Een voorbeeld van een probleem dat deze Solver kan oplossen is: de torens van Hanoi. In 1958 kwam McCarthy met de eerste programmeer taal voor KI: Lisp (list processing). Deze taal wordt, weliswaar in een geëvolueerde vorm, nog steeds gebruikt.

In de KI neemt communicatie van computers in de natuurlijke taal een belangrijke plaats in, zo werden vanaf begin jaren 60 programma’s gebouwd die gericht waren op een natuurlijke taal (Engels). Een van de eerste programma’s in deze richting was STUDENT (geschreven in Lisp), een programma om middelbare school algebra in de vorm van kleine verhalen op te lossen. Semantische

4 “Een poging de wiskunde te funderen in de symbolische logica, vormde het hoogtepunt van het logicistische onderzoeksprogramma. Het werk geldt in de logica als het invloedrijkste na Aristoteles' Organon.“ [30]

netwerken werden in volgende programma's gebruikt als een volgende stap in de richting van het gebruik van natuurlijke talen in computers.

Eind jaren 60 introduceerden Minsky en Papert (MIT Kunstmatige Intelligentie centrum) de term micro-werelden. Hiermee wilden ze aangeven dat de KI zich moet focussen op versimpelde, perfecte versies van de werkelijkheid. De gedachte hierachter was dat dit ook werkt in bijvoorbeeld de natuurkunde, vraagstukken worden makkelijker en beter begrepen als bijvoorbeeld de wrijving buiten beschouwing wordt gelaten. De vorm van deze micro werelden in de KI bestaat vooral uit blokkenwerelden (denk aan Lego steentjes). Minsky en Papert bouwden ook een robot arm die de blokken kon stapelen. Hier opvolgend ontstond SHRDLU, een programma ontwikkeld met Lisp door Terry Winograd rond 1969. Dit programma is de grootste prestatie voortkomend uit het blokkenwereld gedachtegoed. Dit programma kan, binnen de blokkenwereld, communiceren in Engelse zinnen, operaties plannen en uitvoeren. SHRDLU is nog steeds online te vinden, bijvoorbeeld op http://hci.stanford.edu/~winograd/shrdlu/ , hier is meer informatie te verkrijgen en is een demo voorbeeld te vinden van het gebruik van SHRDLU.

2.2 Agenten

Met alle ontwikkelingen op het gebied van KI in het achterhoofd ontwikkelde Joseph Weizenbaum in 1966 de eerste software agent: ELIZA. ELIZA doet zich voor als een psychotherapeut, waarmee je een consult kan houden, en die je het idee geeft dat je een intelligent gesprek hebt. Dit doet het programma door middel van het identificeren van sleutelwoorden in de vragen die het gesteld wordt, waarna gekozen wordt uit een aantal standaard antwoorden. De intelligentie van deze chat bot is erg laag, omdat herhaling van zinnen snel voorkomt

Ondanks de revolutionaire ontwikkeling van deze software agent, werd er in de jaren 70 en 80 weinig verder onderzoek verricht naar agenten. Door de limieten op de rekenkracht in computers en de exponentiële tijd in rekenkracht nodig voor veel problemen konden er alleen voor de simpele, triviale versies van deze problemen oplossingen gevonden worden. Verder zijn er voor bepaalde KI applicaties (leren van taal, 'lezen' van afbeeldingen) enorme hoeveelheden informatie van de wereld waarin wij leven nodig. De informatie die nodig is, is vergelijkbaar met gezond verstand van de mens en in die tijd was het niet mogelijk zo veel informatie op te slaan in één database. Door dit soort zaken ontstond het idee van Moravec's paradox[5]: simpele handelingen voor de mens, zoals zien en lopen, is moeilijker voor een computer dan moeilijke handelingen voor de mens, zoals bewijzen van theorieën en oplossen van geometrie. Nog een tegenslag voor de KI was het Chinese kamer argument van John Searle: Deze redenatie legde een bom onder het doel van de KI om intelligentie in machines te stoppen.

Door al deze zaken lag de focus in deze jaren niet op agenten maar vooral op de analyse van taal (logica), de presentatie van kennis (databases, neurale netwerken etc), op automatisch leren en op computervisie (computers die afbeeldingen kunnen herkennen en lezen).

Dit beeld veranderde pas in de jaren negentig met de komst van het WWW voor het grote publiek. Op dat moment was er ook veel progressie geboekt met het oplossen van de restricties uit de jaren 70-80, wat Oliver McBryan in staat stelde om begin jaren negentig de World Wide Web Worm te bouwen. Deze intelligente agent was gebouwd om alle webpagina's te indexeren in een database,

5 "It is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility." [31]

waarna deze database doorzocht kon worden via sleutelwoorden, de eerste WWW zoekmachine was een feit. Deze agent stimuleerde de bouw van nog meer agenten.

Vanwege de groeiende populariteit van het winkelen via het Web, werd in 1997 Roboshopper gebouwd. Deze winkel assistent agent zocht bepaalde items op het web en vergeleek de prijzen op de verschillende websites. Een ander voorbeeld van een agent in die tijd is de Office Assistant, een agent die de gebruiker van Microsoft Office programma's assisteerde bij het gebruiken van deze programma's. Verder kwamen de Web crawlers op, agenten die zoek machines up to date houden door middel van het speuren naar nieuwe websites en het toevoegen van deze websites aan de zoekmachines.

Al snel bleek dat deze intelligente agenten de toekomst hebben en er werd meer en meer onderzoek naar gedaan. Er werd getracht het hoe en wat van de agenten te vangen in de literatuur. Door de vele verschillende invalshoeken op agenten zijn agenten niet op één manier te definieren. Er is hierdoor alleen een algemeen geaccepteerd beeld van wat agenten zijn, hoe ze werken en wat ze doen. Dit beeld zal in het tweede deel van dit werkstuk aan bod komen.

Deel II – Intelligente agenten

Om een beeld te krijgen van wat agenten zijn, hoe ze werken, hoe ze gebouwd worden en waarvoor ze zijn, wordt in dit deel van het werkstuk de agent ontrafeld. De eigenschappen van een agent komen aan bod, de omgeving van een agent wordt uitgediept en architecturen voor het ontwikkelen van agent-systemen worden besproken.

Hoofdstuk 3 - Definitie van een agent[6]

3.1 Introductie

De noodzaak voor het ontwikkelen van agenten bestaat uit het automatiseren van taken, het reduceren van onplezierig, complex en moeilijk werk, het verpersoonlijken van diensten (en software) , het managen van eisen (zaken) waarvoor veel voorzieningen nodig zijn en dit alles op elk moment van de dag, 24/7.

Omdat agenten voor komen op vele verschillende gebieden en met vele verschillende doeleinden heeft elk vakgebied een eigen definitie van wat agenten zijn. Deze definities zijn vaak toegespitst op het gebruik van agenten in de vakgebieden. Er is in de literatuur geen precieze overeenstemming over de definitie van een agent.

Uit alle definities komt in ieder geval naar voren dat agenten systemen zijn die autonoom kunnen handelen. Michael Wooldridge [37] hanteert de volgende definitie: “Een agent is een computersysteem, dat autonoom kan handelen (doen en denken) in een bepaalde omgeving om zo de toegewezen doelen te halen.” Deze definitie is naast een algemene, ook degene die de essentie van agenten, de autonomie, benoemt. Voor andere definities zie bijlage 2 (Engels).

3.2 Eigenschappen die een agent bezit[7]

Het autonoom handelen ontstaat doordat het gedrag van een agent bepaald wordt door zijn ervaring. De ervaring is gebaseerd op de kennis die de agent meekrijgt en op kennis die door middel van leermechanismen is aangeleerd. Hiernaast hebben agenten nog meer eigenschappen, die vaak gezien worden als de eigenschappen die voor de intelligentie zorgen.

1) **Reactief**
   Een agent heeft een continue interactie met de dynamische omgeving, waardoor er direct (in een korte tijdspan) gehandeld kan worden als een omgeving verandert.

2) **Proactief**
   Een agent is doel georiënteerd en kan zelf initiatief nemen om de doelen te halen, zonder dat er interventie is van mensen of andere agenten. Met andere woorden, de agent herkent zelf mogelijkheden in een dynamische omgeving en reageert hier op.

6 Hoofdstuk 3 is onder andere gebaseerd op [1]
7 Sectie 3.2 is onder andere gebaseerd op [5], [37], [38]

3) **Sociaal**
   Ook is een agent sociaal. Onder deze eigenschap wordt verstaan dat een agent interactie kan hebben met mensen en/of andere agenten via coöperatie, coördinatie en onderhandelen.

Van coöperatie is sprake als meerdere agenten samenwerken om een gezamenlijk doel te bereiken. Agenten maken gebruik van deze eigenschap op het moment dat een agent alleen het doel niet kan bereiken of als er met meerdere agenten een beter resultaat te behalen is.
Coördinatie vindt plaats in situaties waarbij meerdere agenten, met eigen doelen, eenzelfde voorziening moeten gebruiken om hun doel te bereiken. Oftewel, de onderlinge afhankelijkheden tussen activiteiten van de agenten moeten afgestemd worden op elkaar, om elkaar niet in de weg te lopen.

Verder hebben agenten nog de mogelijkheid om met elkaar te onderhandelen. In deze onderhandelingen speelt geven en nemen een grote rol, er moet uiteindelijk een compromis tussen de agenten worden bereikt.

Interessant aan deze manier van denken over agenten is dat deze aangeeft dat de mens ook een agent is.

Naast de hierboven genoemde eigenschappen die alle agenten hebben, zijn er ook andere eigenschappen die agenten kunnen bezitten, waarvan hieronder enkele voorbeelden gegeven worden:

1) **Mobiliteit**
   Met deze eigenschap is een agent niet gebonden aan een platform of machine waarop het draait. Mobiliteit zorgt er bijvoorbeeld voor dat een agent zich door een netwerk kan bewegen. Het neemt de code om iets uit te voeren mee naar de voorziening of host waarop het uitgevoerd moet worden. Dit scheelt bandbreedte en heeft ook als voordeel dat agenten met deze eigenschap niet afhankelijk zijn van een continue netwerk connectie. Juist hierdoor zijn deze agenten geschikt voor mobiele (wireless) computing systemen, systemen waarin mobiele netwerk hosts voorkomen. Zie ook in het laatste hoofdstuk van deel II: Mobiele Agenten.

2) **Ethisch**
   Een agent met deze eigenschap zal nooit moedwillig een andere agent van verkeerde informatie voorzien.

3) **Welwillendheid**
   Deze eigenschap komt neer op het bereid zijn om te reageren op hulp aanvragen van andere agenten.

4) **Rationeel**
   Agenten volgen dat pad naar het doel dat de ‘prestatie’ van de agent maximaliseert.

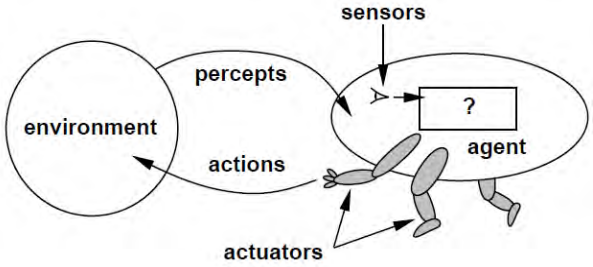
5) **Persistent**
   Dit houdt in dat een agent doorgaat met het uitvoeren van een proces totdat het beoogde resultaat is behaald.

6) **Redeneren**
Voordat een agent een taak uitvoert zal het bekijken welke acties het kan nemen om een bepaald resultaat te behalen oftewel, de keuze voor een actie ligt dus bij wat de uitkomst moet zijn.

De genoemde eigenschappen kunnen gezien worden als persoonlijkheden van een agent. Deze eigenschappen laten alleen niet zien hoe een agent omgaat met input en output en hoe agenten werken zoals ze werken. Om dit te laten zien dient er gekeken worden naar de structuur van een agent.

Hoofdstuk 4 - Structuur van een agent[8]

Een definitie van wat een agent is die dieper ingaat op de structuur van een agent komt van Russell en Norvig [20]. Zij stellen dat een agent een entiteit is die gezien kan worden als iets dat z'n omgeving kan waarnemen door middel van sensoren en actie kan ondernemen aan de hand van de toestand van de *omgeving* door middel van zogenaamde *actuatoren*. Voorbeelden van actuatoren zijn een stuur en een rempedaal in een auto, een weergave van vragen op een scherm, uitvoeren van testen in een diagnostisch systeem en een beweegbare robotarm. Een andere stelling van Russell en Norvig is een Agent = Agent Programma + Agent-architectuur



Figuur 2: De definitie van een agent-systeem volgens Russell en Norvig. [20]

In deze stelling is de agent de architectuur het computer-systeem, een computer of speciale hardware, met bepaalde besturing-software, waarop een agent programma draait. Het bepaalt het gedrag van een agent in een bepaalde omgeving: het beeld van de omgeving, de input via de sensoren van de agent, de zogenaamde *perceptie van een agent* genoemd, worden door de architectuur verwerkt en aangeboden aan het agent programma, waarna de architectuur het programma draait. Daarna zullen de uitkomsten van het programma, beslissingen van acties die genomen moeten worden, door de architectuur doorgegeven worden aan de actuatoren (uitvoerders) van de agent.

In tegenstelling tot Russell en Norvig wordt in het algemeen de agent-architectuur gezien als een blueprint om een agent, met eerder genoemde 'eigenschappen, te bouwen. Pattie Maes (1991) (gerenommeerde onderzoekster op dit gebied) definitie van een agent-architectuur is: "Een bepaalde methodiek om agenten te bouwen die aangeeft hoe de agent verdeeld wordt in een set van modules, en hoe deze modules interactie kunnen hebben. De set van modules en de interactie moet ervoor zorgen dat er een antwoord is op de vraag hoe de input van de sensoren en de huidige interne toestand van de agent de acties en de toekomstige interne toestanden bepaalt. De architectuur omvat technieken en algoritmes om deze methodiek te ondersteunen."
De architectuur van een agent bestaat dus uit interne structuren, waarbij het voor de meeste belangrijk is dat deze in real time uitgevoerd worden. Zo is er een structuur die ervoor zorgt dat een

8 Hoofdstuk 4 is onder andere gebaseerd op [20], [25]

agent kennisrepresentatie heeft. Wat erop neer komt dat een agent feiten en regels over een bepaald onderwerp bezit, en zo een interpreteerbare beschrijving van de werkelijkheid heeft. Verder is er een interne structuur voor het representeren en behalen van gestelde doelen. Deze zorgt onder andere voor de manier van redeneren met betrekking tot de planning van acties, de programma's die gebruikt worden en de acties zelf. Zo zijn er algoritmes voor het automatisch construeren en uitvoeren van simpele commando's om de high-level doelen te bereiken. Op dit gebied wordt veel onderzoek gedaan om talen te ontwikkelen voor dynamische systemen en om algoritmes te ontwikkelen die voor synthese van de mogelijke acties moeten zorgen. Ook omvat de architectuur structuren die voor interactie met de omgeving zorgen en die met onregelmatigheden kunnen omgaan.

Structuren die niet alle agenten bezitten, maar bij de architectuur kunnen horen, zijn degenen die voor coördinatie en collaboratie met andere agenten zorgen,en die voor menselijk gedrag en zelfs emotie kunnen zorgen. Hieronder vallen ook structuren voor natuurlijke taal (bijvoorbeeld spraak) en leren & aanpassen. Dit laatste houdt in dat een agent zich ontwikkelt door middel van ervaring en dat een agent regels behorende bij een bepaald onderwerp leert uit een grote hoeveelheid data.

De grootste uitdaging op het gebied van de agent-architectuur is het ontwikkelen van de geschikte architectuur bij een bepaalde omgeving. Immers, de omgeving bepaalt de manier waarop een agent ontworpen wordt en welke interne structuren deze nodig heeft.

Linden [13] stelt het volgende voor met betrekking tot het ontwerpen van een architectuur:

1) **Karakteriseer** de benodigdheden van de agent aan de hand van de kwesties waar deze agent voor gebruikt gaat worden. Dus wat heeft de agent allemaal nodig en op welke manieren moet het omgaan met zaken als redeneren, beperkingen op beschikbare middelen, kennis, autonomie, user interactie, control flow (volgorde van uitvoeren van instructies).

2) **Evalueer** bestaande architecturen die het dichtst in de buurt van de lijst met benodigdheden komen en kies hieruit de meest geschikte.

Naast tijdsbesparing zijn er nog meer redenen om dit proces te verkiezen boven het zelf ontwikkelen van een architectuur. Het bouwen van een architectuur waarin de juiste mix van benodigdheden (interne structuren) ook voldoende compatibiliteit met elkaar hebben, blijkt een kunst te zijn. Om real-time, geschikt, intelligent gedrag van een agent te ondersteunen moeten de interne structuren perfect geïntegreerd worden tot een geheel, anders zal de werking van een agent niet het juiste resultaat opleveren en zal de agent dus inferieur zijn.

In dit deel van het werkstuk ligt de focus op de single-agent. Hierbij horen architecturen die gericht zijn op één agent: single-agent-architectuur. Deze architecturen richten zich op het interne, dus perceptie – beslissingen nemen – acties ondernemen, van een agent en op de interactie van de agent met de omgeving. Er blijken in de literatuur veel verschillende architecturen te zijn welke op hun beurt weer ingedeeld zijn in soorten.

Hoofdstuk 5 - Omgeving van een agent[9]

5.1 Introductie

Een belangrijk aspect bij het kiezen van een juiste architectuur en agent-programma (later meer hierover) en dus voor het ontwerpen van een agent, is de omgeving waarin de agent gesitueerd wordt. Pas als agenten bekend zijn met hun omgeving (er kennis over bezitten) en als agenten eigenschappen bezitten die continu voor meer kennis van de omgeving zorgen, kunnen ze optimaal functioneren.

Een agent-omgeving omvat alles buiten de agent dat met de agent interactie heeft. Deze omgeving bestaat uit een set van taken, waarbij een taak een probleem of doel is, die de agent moet oplossen of moet behalen. Belangrijk voor de agent is de toestand van de omgeving, die via de perceptie van een agent 'gelezen' wordt. Aan de hand van de perceptie van de huidige omgevingstoestand wordt vervolgens een beslissingen genomen tot een geschikte actie, waarna de agent deze actie zal uitvoeren. De actie wordt uitgevoerd op de omgeving, waarna de omgeving verandert en er een nieuwe situatie voor de agent ontstaat.

5.2 Omgevingseigenschappen

Hoe de omgeving is opgebouwd, welke eigenschappen deze heeft, vanuit het oogpunt van de agent is belangrijk voor de keuze van het juiste ontwerp van de agent. In het duiden van de opbouw is het van belang om van te voren de precieze omgeving aan te geven. Er is immers een verschil tussen of de agent naar het geheel moet kijken (bijv. een toernooi) of naar een gedeelte van dat geheel (bijv. een wedstrijd in het toernooi). Als deze omgeving gedefinieerd is, kan de opbouw van de omgeving geduid worden. Zo is een omgeving:

1) **Volledig of gedeeltelijk waarneembaar**
   Als de sensoren van de agent de gehele omgevingstoestand op elk moment kunnen waarnemen wordt de omgeving volledig waarneembaar genoemd. Is dit niet het geval dan zal de agent schattingen moeten doen over de op dat moment niet waarneembare delen van de omgevingstoestand.

2) **Deterministisch, stochastisch of strategisch**
   Een omgeving wordt omschreven als deterministisch als de volgende omgevingstoestand volledig afhankelijk is van de huidige toestand en van de acties die de agent maakt op de huidige toestand. Een strategische omgeving is deterministisch met het verschil dat het niet om de acties van de agent zelf gaat, maar om de acties van andere agenten. Een omgeving waarin de volgende toestand onafhankelijk is van de huidige toestand en onafhankelijk van de acties van de agent(en) wordt gezien als stochastisch. In dit geval zal een agent dus eigenschappen moeten bezitten die schattingen maakt over veranderingen in de omgeving.

3) **Episodisch (toevallig) of sequentieel (opeenvolgend)**
   Als toekomstige acties afhankelijk kunnen zijn van de huidige acties die genomen worden, wordt de omgeving sequentieel genoemd. Het is duidelijk dat in dit geval een agent vooruit zal moeten plannen. De omgeving wordt episodisch genoemd als de huidige te nemen acties niet afhankelijk zijn van vorige acties, en dus ook geen invloed hebben op toekomstige acties.

9 Hoofdstuk 5 is onder andere gebaseerd op [5], [20]

4) **Statisch of dynamisch**
Als gedurende het nadenken en beslissen van een agent over de volgende actie de omgeving niet verandert, dan wordt de omgeving statisch genoemd. De omgeving verandert dus alleen door de acties van de agent. Hier tegenover staat de dynamische, dus constant veranderende, omgeving. Omdat deze omgeving continu verandert niet door toedoen van de agent, zal de agent zich continu moeten aanpassen om nog steeds de juiste beslissingen te nemen. Om dit te bereiken kan een agent in overleg gaan met de omgeving, anticiperen op de veranderingen in de omgeving of heel snel beslissingen nemen.

5) **Discreet of continu**
Of een omgeving aangemerkt kan worden als discreet of continu ligt aan het aantal toestanden dat deze kan aannemen. Als dit aantal (bijna) oneindig groot is, wordt de omgeving continu genoemd. Anders is de omgeving discreet.

6) **Single-agent of multi-agent**
In het geval van een single-agent-omgeving is er slechts één agent werkend. In een multi-agent-omgeving werken meerdere agenten samen om de doelen te behalen.

Het Web is een voorbeeld van een populaire agent-omgeving. De eigenschappen van deze omgeving worden uitgezet in bijlage 3. In *Figuur 3* is een overzichtje gegeven van de eigenschappen van een aantal andere agent-omgevingen.

	Crossword puzzle	Chess	Back-gammon	Internet shopping	Taxi	Part-picking robot
Fully observable	yes	yes	yes	no	no	no
Deterministic	yes	strat.	no	(yes)	no	no
Episodic	no	no	no	no	no	yes
Static	yes	semi	yes	semi	no	no
Discrete	yes	yes	yes	yes	no	no
Single agent	yes	no	no	no	no	yes

Figuur 3: Een overzicht van de eigenschappen van enkele agent-omgevingen. [2]

Zo is er een agent die kruiswoordpuzzels maakt (dus in een kruiswoordpuzzel omgeving zit), die schaak speelt, die backgammon speelt, en een die shopt op het internet.
Onder **Taxi** wordt een geautomatiseerde taxi chauffeur agent verstaan, die klanten van A naar B brengt zonder dat daar een menselijke chauffeur aan te pas komt. Met **Part-picking robot** wordt een agent bedoeld die van een lopende band componenten pakt en ze in een bak met dezelfde soort componenten legt.

5.3 Agent-omgevingsmanagement

Uit het Web voorbeeld (zie bijlage 3) komt naar voren dat een zeer complexe, dynamische omgeving een agent noodzaakt om slimmer, beter en efficiënter te worden om te overleven (bruikbaar te blijven). Agent-omgevingsmanagement is een hulpmiddel, geredeneerd vanuit de omgeving, om ervoor te zorgen dat agenten effectief blijven. Omgevingsmanagement probeert een omgeving te creëren waarin de agent effectief kan werken. Onderdeel van deze management is terraforming, dat probeert veranderingen in de omgeving aan te brengen waardoor agenten effectief blijven werken in de omgeving.

Hoofdstuk 6 - Agent-programma

Zoals eerder genoemd gaat het bij het kiezen van de juiste architectuur om het inpassen van de agent in een bepaalde omgeving. Datgene van de omgeving wat de agent op elk moment binnenkrijgt via de sensoren, wordt de perceptie van de agent genoemd. De variabelen van de input vormen de toestanden die de omgeving kan aannemen (s_i). De toestandsruimte wordt beschreven als $\mathbf{S = \{ s_1, s_2, s_3 \ldots\}}$. In elke omgevingstoestand moet de agent een optimale actie vinden en uitvoeren. De acties die genomen kunnen worden, worden beschreven in $\mathbf{A = \{ a_1, a_2, a_3 \ldots\}}$. De agent-functie zorgt ervoor dat elke opeenvolging van percepties ingedeeld wordt bij een bepaalde 'beste' actie. Hiermee beschrijft de agent-functie dus het gedrag van de agent en daarmee de agent.

In formule vorm ziet de agent-functie er als volgt uit:

> **Actie: S* -> A.**
>
> Opeenvolgingen van omgevingstoestanden (**S***: meerdere **S**) worden ingedeeld (engels: mapping) bij acties.

Ook de omgeving kan in formule vorm worden uitgedrukt:

> **Omgeving: S x A -> ρ(S)**
>
> De huidige toestand van de omgeving en een actie kunnen een set, **ρ(S)**, van nieuwe omgevingstoestanden bereiken. Als deze set continu maar één mogelijkheid bevat, is de omgeving deterministisch. Immers, dan is precies te zeggen hoe de omgeving zich gedraagt

De interactie tussen agent en de omgeving wordt als volgt uitgedrukt:

> **Geschiedenis (h): $s_0 -a_0-> s_1 -a_1-> s_2 -a_2-> s_3 -a_3-> \ldots$**
>
> Met $\mathbf{S_0}$ de begintoestand van de omgeving.
>
> Uit de set van alle mogelijke interactie geschiedenissen blijkt wat het karakteristieke gedrag van de agent in de omgeving is.

Om de agent-functie te implementeren in een agent, moet er een tabel worden gemaakt van alle mogelijke **S*** met alle daarbij behorende A. Omdat dit in veel situaties niet te doen is, vanwege de zeer grote toestandsruimtes, wordt er vaak gebruik gemaakt van een andere implementatie van de indeling van **S* -> A**. Een implementatie van de agent-functie wordt daarom een agent-programma genoemd. Waar de agent-functie een **S*** (opeenvolging van toestanden) als input heeft, heeft zo'n

programma een enkele input **S**. Om dit verschil op te lossen wordt in het programma (in de agent) een interne toestand van de omgeving bijgehouden.

Percept x	Action z
1.0	1.000000000000000
1.1	1.048808848170152
1.2	1.095445115010332
1.3	1.140175425099138
1.4	1.183215956619923
1.5	1.224744871391589
1.6	1.264911064067352
1.7	1.303840481040530
1.8	1.341640786499874
1.9	1.378404875209022
⋮	⋮

function SQRT(x)
  $z \leftarrow 1.0$ /* *initial guess* */
  repeat until $z^2 - x < 10^{-15}$
    $z \leftarrow z - (z^2 - x)/(2z)$
  end
  return z

Figuur 4: Het vinden van de wortel z van een getal x, met links een tabel van alle mogelijke getallen en de daarbij behorende uitkomst en rechts de methode van Newton. [20]

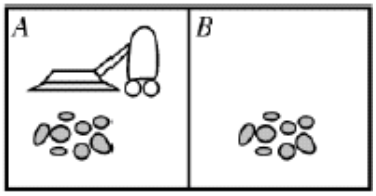
Een simpel voorbeeld van een agent-functie is te vinden in *Figuur 4*. Bovenstaande indelingen van **S* -> A**, het vinden van de wortel van een getal, zijn beide agent-programma's. Het is duidelijk dat de tabel oneindig groot is en dus dat de compacte Methode van Newton aan de rechterkant de voorkeur heeft. Het ontwerpen van een nauwkeurig en compact agent-programma moet ook gebeuren in algemenere situaties: want daar heeft een agent te maken met oneindig veel verschillende taken in een omgeving die oneindig veel toestanden kan aannemen.

Hoofdstuk 7 - Performance measures en PEAS

Uit het feit dat de agent-functie uit alle mogelijkheden van **S* -> A** de ´beste´ volgende actie kiest, blijkt dat agent-functies rationeel zijn. Hierdoor zal op zijn beurt een agent altijd de juiste actie kiezen. Welke actie de juiste is, wordt bepaald door het succes die de agent behaalt met de actie. Om dit succes objectief te bepalen moeten er numerieke 'performance measures' (metingen van de prestatie) voor het gedrag van de agent zijn die de doelen van de agent representeren. Het is beter om performance measures te ontwikkelen die aangeven wat een gebruiker wil in de omgeving in plaats van wat het gedrag van een agent in de omgeving moet zijn. De agent moet uiteindelijk deze performance measures proberen te optimaliseren, maar het is aan het agent-programma om te bepalen wat de beste manier is om dit te bereiken.

Het volgende bekende voorbeeld uit Russell en Norvig [20] vat dit laatst genoemde samen en verduidelijkt waar nodig.

Stel er is een stofzuiger agent die een vloer, die is onderverdeeld in locaties,schoon moet maken en houden, zie *Figuur 5* . De volgende gegevens zijn van toepassing:



Figuur 5: Een stofzuiger agent

- Er zijn twee locaties: **A** en **B**.
- De perceptie van de agent bestaat uit: **de locatie** en **de vuilgraad** van de locatie (binair is dit: schoon of vuil).
- Acties die de agent kan uitvoeren zijn: **'Ga naar links'** (uitvoerbaar als de agent in B zit), **'ga naar rechts'** (uitvoerbaar als de agent in A zit), **'stofzuig'**, **'doe niks'.**

De agent-functie ziet er dan als volgt uit:

Opeenvolging van de percepties (S*)	Acties (A)
[A,Schoon]	Ga naar rechts
[A,Vuil]	Stofzuig
[B,Schoon]	Ga naar links
[B,Vuil]	Stofzuig
[A,Schoon],[A,Schoon]	Ga naar rechts
[A,Schoon],[A,Vuil]	Stofzuig
…	…
[A,Schoon],[A,Schoon],[A,Schoon]	Ga naar rechts
[A,Schoon],[A,Schoon],[A,Vuil]	Stofzuig
…	…

Figuur 6: De agent-functie van de stofzuiger agent.

Een simpel agent-programma met deze gegevens zou gebaseerd kunnen zijn op:

> *Als de huidige locatie (het vlak waar de agent zich op bevindt) vuil is, zuig dan het vuil op. Anders, ga naar het andere vlak. (Merk op: IF-THEN-ELSE)*

Stel nu dat er een performance measure is die inhoudt dat elke verplaatsing een boete oplevert. Een uitbreiding op het agent-programma zegt dan: *doe geen actie op het moment dat de agent weet dat elk vlak schoon is*. Uiteraard, als gegeven is dat de vlakken weer vies kunnen worden, zal de agent eens in de zoveel tijd de locaties moeten checken (maar niet continu). Er zou dan een leermechanisme ingebouwd moeten worden die een optimale tijd vindt opdat de agent een vlak gaat checken.
Vergelijk bij een stofzuiger agent een performance measure bij '*de gebruiker wil een schone vloer*' (bijvoorbeeld een beloning per minuut dat de vloer schoon is) met een performance measure die vanuit het gedrag van een agent redeneert '*de hoeveelheid stof die de agent heeft opgezogen*'. Bij dit laatste geval is het meest optimale voor een agent om stof te blijven opzuigen, oftewel, om continu de zak met stof te legen op de vloer en weer van voor af aan te beginnen.

De 'juiste' actie (rationele actie) die de agent neemt, gegeven de geschiedenis (h), de a priori kennis en de verkregen, aangeleerde, kennis, zal de performance measures maximaliseren. De hoeveelheid a priori kennis, kennis meegegeven door de ontwikkelaar, die de agent gebruikt bepaalt de autonomie van een agent. Doel van het ontwikkelen van een agent is om zo min mogelijk kennis mee te geven. Naast het feit dat dit veel dure kennis van deskundigen scheelt, zijn deze agenten ook beter in staat zich aan te passen aan veranderende of andere omgevingen. Het is duidelijk dat deze autonome agenten, die zelf hun kennis moeten opdoen, een stuk lastiger te ontwikkelen zijn dan de probleem specifieke agenten met alle mogelijke kennis ingebouwd.

Voordat begonnen wordt met het ontwikkelen van een agent is het belangrijk om de zogenaamde taak omgeving zo volledig mogelijk te definiëren. Het is zo dat de agent-omgeving uit een aantal taken bestaat; problemen of doelen die een agent moet aanpakken. De agent-omgeving met openstaande taken wordt daarom ook wel de taak omgeving genoemd. Kortweg komt dit neer op het vaststellen van de input, met name hoe deze te verkrijgen is (welke sensors zijn nodig) en het vaststellen van de output, met name hoe acties te verwezenlijken zijn (welke actuatoren zijn hiervoor nodig). En uiteraard mag een beschrijving (overzicht waaruit de omgeving bestaat) van de omgeving niet ontbreken. De algemene methode om een taak omgeving te definiëren, de specificatie van het probleem, is om gebruik te maken van *PEAS*. Wat staat voor Performance measures (metingen van de prestatie), Environment (agent-omgeving), Actuators (actuatoren, uitvoerders), Sensors

(sensoren). Het gebruik van deze methode zorgt voor een zeer bondige en overzichtelijke specificatie van het probleem, zoals hieronder te zien is.

Wat volgt is een simpele PEAS specificatie voor een 'part-picking robot agent' , een agent-systeem die autonoom, op een intelligente manier, van een lopende band componenten pakt en ze in een bak met dezelfde soort componenten legt (in het kort: sorteren). In de agent zijn de sensoren voor het bepalen van het soort component en de actuatoren zijn voor het oppakken en indelen van de componenten. Voor een agent-systeem met een uitgebreid PEAS schema zie ook sectie 11.2.

Part-picking robot agent:	
1) Performance Measures:	Percentage van componenten in de juiste bakken ingedeeld, snelheid, …
2) Environment:	Een lopende band met de te sorteren componenten, bakken waarin de componenten ingedeeld moeten worden, …
3) Actuators:	Een robot arm (arm – gewricht – hand), …
4) Sensors:	Camera's, sonar sensoren, . . .

Figuur 7: PEAS overzicht van een 'part-picking robot agent'.

Alles tot nu toe besproken, PEAS, de omgeving, het agent-programma etc. worden samengevoegd om de agent te ontwikkelen. Voor het daadwerkelijk bouwen van de agent-programma 's in de agent onderscheiden Russell en Norvig [20] vijf typen agenten, door o.a. J. Hsu [10] aangemerkt als abstracte architecturen.

Hoofdstuk 8 - Abstracte architecturen

8.1 Simple reflex agents (simpele reflex agenten)

De minst algemene, of degene met de minste intelligentie, van de vijf architecturen is slechts gebaseerd op de conditie-actie regel (beter bekend als if-then). Omdat via deze regel alleen vanuit de huidige perceptie van de toestandsruimte, dus niet de geschiedenis, de juiste actie wordt geselecteerd, is deze architectuur alleen te gebruiken in een *volledig waarneembare omgeving*.

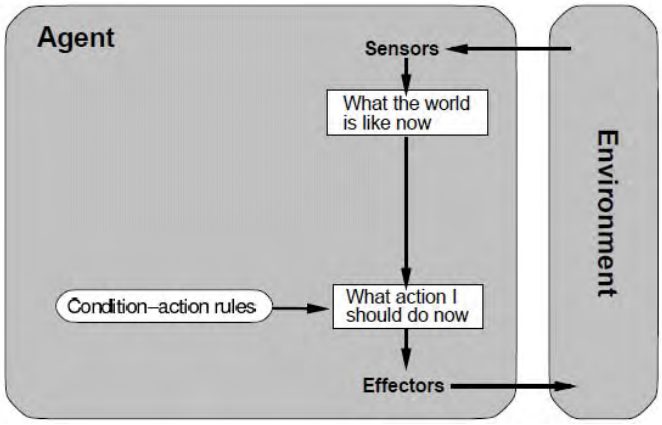
Een simpele reflex agent-programma voorbeeld: de stofzuiger agent.

```
function REFLEX-VACUUM-AGENT([location,status]) returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

Figuur 8: Agent-programma voor de stofzuiger agent uit hoofdstuk 7.

De huidige perceptie van de toestandsruimte wordt gekoppeld aan een conditie uit de lijst van a priori ('ingebouwde') condities en eventuele aangeleerde condities en daarmee aan een actie.

De algemene structuur van deze agenten ziet er als volgt uit:




```
function SIMPLE-REFLEX-AGENT(percept) returns action
  static: rules, a set of condition-action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  return action
```

Figuur 9: De simpele reflex agent, schematisch (links) en in een agent-programma beschreven (rechts).

In *Figuur 9* is links het schematisch diagram te zien van de agent die in deze sectie beschreven is. Het blokje *What the World is like now* stelt de huidige perceptie voor.

Rechts in *Figuur 9* is een agent-programma voor dit type agent weergegeven. Het agent-programma heeft een *INTERPRET-INPUT* functie die de perceptie van de omgevingstoestand (percept) vertaalt en beschrijft voor de agent. *RULE-MATCH* functie die deze toestand koppelt aan een conditie en de *RULE-ACTION* functie die de juiste actie koppelt aan deze conditie. In het voorbeeld van de stofzuiger agent zijn deze laatste twee functies in een functie samengevoegd.

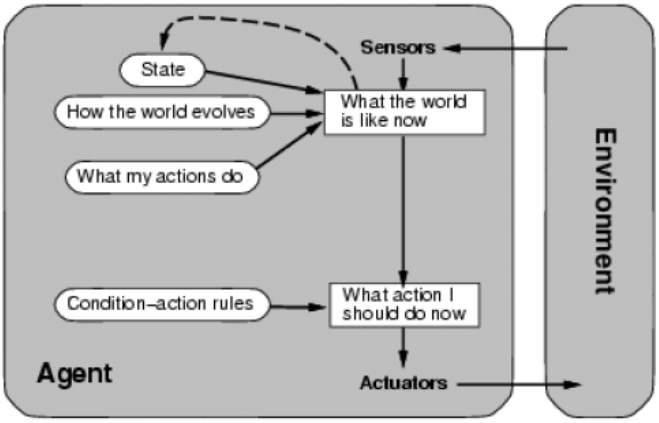
8.2 Model-based reflex agents with state (op model gebaseerde reflex agenten met interne toestand)

Een volledig waarneembare omgeving is in de meeste situaties niet haalbaar. Toch is het op een bepaald moment niet waarneembare deel van de omgeving van belang voor de agent om een juiste beslissing te nemen, aangezien dit deel van de omgeving ook verandert. *Bijvoorbeeld: in je achteruitkijkspiegel zie je een auto je tegemoet komt rijden met een rechterknipperlicht aan. Op het moment dat je niet meer kijkt betekent dit niet dat de auto niet meer is of dat de auto niks doet.*

Om de op het moment niet waarneembare delen van de omgeving op te vangen wordt er een interne omgevingstoestand bijgehouden van de gehele omgevingstoestand. Deze interne toestand wordt geüpdate met behulp van een model van de werking van de wereld. Dit model is gebaseerd op de geschiedenis van de percepties van de omgevingstoestanden, informatie over hoe de omgeving evolueert onafhankelijk van de agent en wat de invloed van de acties van de agent op de omgeving is. *In het voorbeeld van de auto in de achteruitkijkspiegel zal het model de auto, die je in de vorige perceptie van de omgeving had gezien, rechts laten afslaan, immers het model weet dat als een auto een rechterknipperlicht aan heeft dat deze, ideaal gezien, rechtsaf gaat. Dit is dan gebeurd in het op dat moment niet waarneembare deel van de omgeving.*

De keuze van de agent voor de juiste actie gaat vervolgens op dezelfde manier als in de simpele reflex agent.

De algemene structuur van deze agenten ziet er als volgt uit:



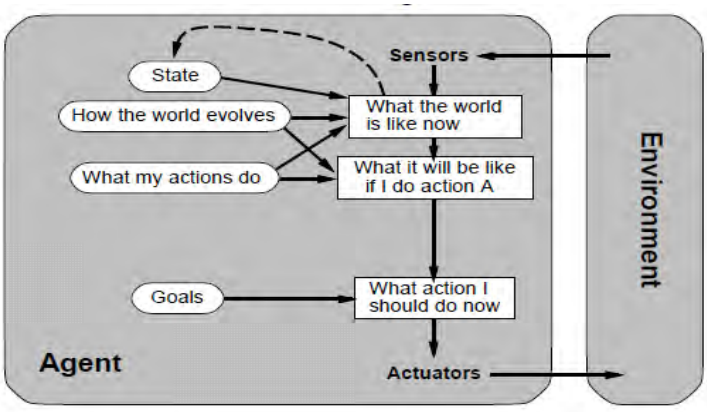

```
function REFLEX-AGENT-WITH-STATE(percept) returns an action
    static:  state,  a description of the current world state
             rules,  a set of condition-action rules
             action,  the most recent action, initially none

    state <-- UPDATE_INPUT(state, action, percept)
    rule <-- RULE_MATCH(state, rules)
    action <-- RULE_ACTION[rule]
    return action
```

Figuur 10: De op model gebaseerde reflex agent met interne toestand, schematisch (links) en in een agent-programma beschreven (rechts).

Met in het schema in *Figuur 10*: *State* - de interne omgevingstoestand gebaseerd op de perceptie geschiedenis, *How the World evolves* - de kennis over de evolutie van dingen in de omgeving, *What my actions do* - de kennis over de evolutie van de omgeving bij bepaalde acties. Het bijbehorende agent-programma heeft als toevoeging *UPDATE_INPUT* functie, die de interne omgevingstoestand update als eerder beschreven.

8.3 Goal-based agents (op doelen gebaseerde agenten)

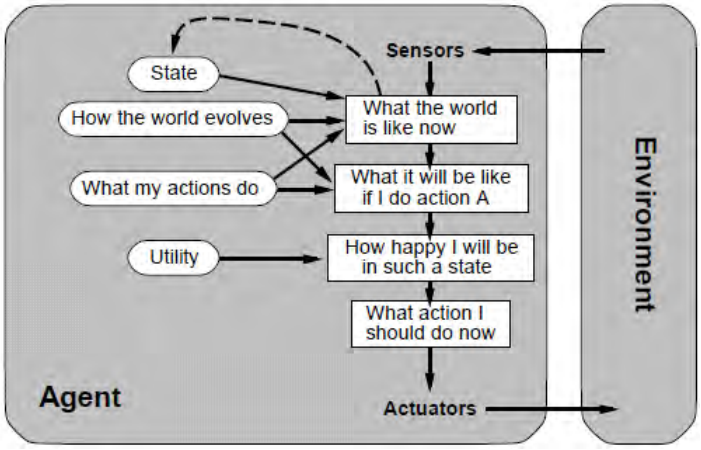
Een interne omgevingstoestand bijhouden is adequaat genoeg voor beslissingen in simpele omgevingen. Een model-based reflex agent die van A naar B wil en onderweg aankomt bij een kruispunt, kan worden voorgeprogrammeerd om dan linksaf te gaan. Maar in situaties waarin niet bekend is dat dit een weg is naar de eindbestemming kan dit niet voorgeprogrammeerd worden. Om dit op te lossen worden er doelen toegevoegd aan agenten. Dit soort agenten zullen dan die acties kiezen waarmee ze de doelen bereiken. Een agent-programma kan de mogelijke acties en gevolgen nagaan en hieruit een kiezen die het doel bereikt. In complexe omgevingen waarbij vele acties nodig zijn alvorens de agent de doelen kan bereiken gebruikt de agent redenering in de vorm van Search en Planning om de juiste reeks van acties te vinden die de doelen bereiken. Search en Planning (Zoek op engelse wikipedia: automated planning and scheduling) moet zorgen voor oplossingen in complexe, multidimensionale en dynamische omgevingen. In een stochastische omgeving komt Search en Planning neer op policy iteratie in Markov Decision Processes (MDP) of zelfs in gedeeltelijk waarneembare MDP's.



Figuur 11: De op doelen gebaseerde agent in een schematisch diagram. In deze agent is de *Condition-action rules* vervangen door *Goals (doelen)*, wat resulteert in een hogere intelligentie.

Het simpele voorgeprogrammeerde if-then in de model-based agenten is dus vervangen door doelen georiënteerd beslissingen nemen. Door de manier waarop een op doelen gebaseerde agent werkt, lijkt deze in sommige gevallen minder efficiënt te zijn dan de model gebaseerde agent. Als bij de voorligger van een auto de remlichten aangaan zal de op model gebaseerde agent remmen. De op doelen georiënteerde agent zal beredeneren dat de voorlichter aan het afremmen is, en omdat een van de doelen van de agent is om andere auto's niet te raken, zal het ook gaan remmen. Deze ogenschijnlijke inefficiëntie wordt volledig gecompenseerd door de flexibiliteit van de op doelen gebaseerde agent: als de agent van A naar een nieuwe bestemming C wil in plaats van naar B hoeft in deze Goal-based agent alleen de eindbestemming aangepast te worden (B wordt C). In een model-based agent zal de hele reeks van acties opnieuw geprogrammeerd moeten worden. De flexibiliteit zit hem dus in de representatie van de kennis, die in de goal-based agent telkens beredeneert wordt en die in de model-based agent helemaal uitgeschreven is door de condities-acties.

8.4 Utility-based agents (utiliteit gebaseerde agenten)

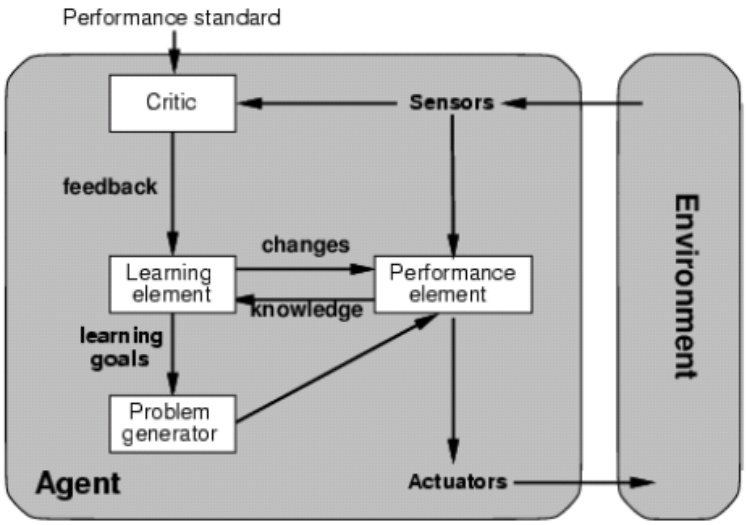


Figuur 12: De utiliteit gebaseerde agent in een schematisch diagram. Waarin doelen vervangen worden door utiliteit, een stempel voor belangrijkheid van mogelijke toestanden en doelen.

Doel gebaseerde agenten (sectie 8.3) kijken alleen naar wat ze moeten bereiken, en niet zozeer naar hoe. Elke acties reeks die leidt tot het gestelde doel kan gekozen worden. Oftewel, een doel gebaseerde agent kijkt naar of een bepaalde toestand, bereikt door de acties reeks, gunstig is voor de agent of niet, wat dus een binaire stempel is. Het gedrag van een agent kan van veel hogere kwaliteit worden als er een functie is die de ene toestand (of toestand reeks) gunstiger inschat voor de agent dan de andere toestand (toestand reeks). Dit is de *utiliteitsfunctie*: een functie die een toestand aan een getal koppelt die de 'blijheid' van een agent beschrijft. De toestand met de hoogste utiliteit heeft de voorkeur voor een utiliteit gebaseerde agent, waarin de doelen dus vervangen zijn (zie *Figuur 12*). Ten opzichte van doelen brengt de utiliteitsfunctie uitkomst bij conflicterende doelen (zoals snelheid en veiligheid). De functie geeft aan wanneer het ene doel de voorkeur krijgt boven de andere, wat dus in elke toestand anders kan zijn. Ook bij het behalen van nog te kiezen, onzekere, doelen kan de utiliteitsfunctie uitkomst bieden. De succeskans van een doel wordt afgewogen tegen de belangrijkheid van het doel (hoe 'blij' wordt de agent hiervan). Bordspellen lenen zich uitstekend voor dit soort agenten.

8.5 Learning agents (lerende agenten)

Alle genoemde agenten uit hoofdstuk 8 kunnen uitgebreid worden met een *leermechanisme* zoals hieronder weergeven. Zo'n mechanisme zorgt ervoor dat agenten continu zichzelf verbeteren en zo meer autonoom worden, wat ze ook geschikt maakt om in nieuwe omgevingen te opereren.



Figuur 13: De lerende agent in een schematisch diagram. Elk van de eerder genoemde agenten kunnen omgevormd worden naar een lerende agent. De originele agent is het *Performance element*, en de rest van het schema vormt het leermechanisme.

In dit schema is het *Performance element* de gehele agent zoals die eerder is beschreven, dus het proces voor het kiezen van de beste actie. Het *learning element* is er om verbeteringen (toevoegingen, aanpassingen) aan te brengen in het Performance element. Het learning element gebruikt de feedback van *critic*, die informatie bevat over hoe goed de agent werkt. *Problem generator* is een generator verantwoordelijk voor het voorstellen van verkennende acties. Dit zijn acties die een gewone agent niet zou doen, maar juist hierdoor voor nieuwe en informatieve ervaringen en kennis zorgt.

Hoofdstuk 9 - Concrete architecturen[10]

9.1 Introductie

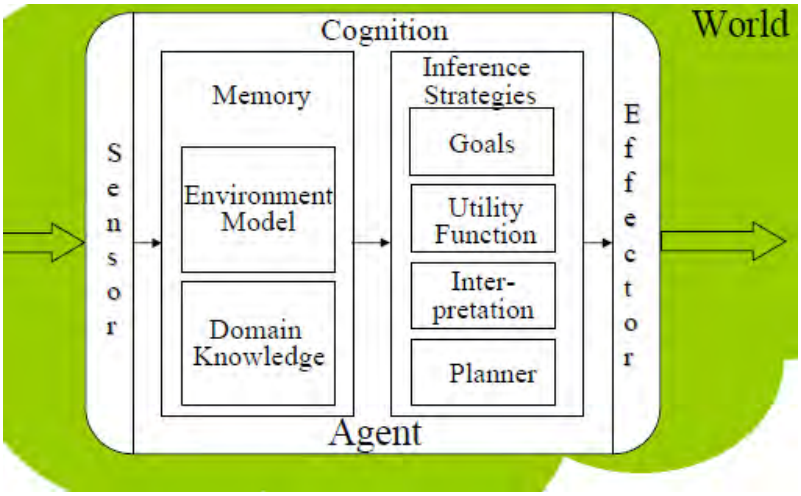
De genoemde abstracte architecturen laten globaal zien hoe een agent werkt, hoe agent-programma's geïmplementeerd kunnen worden, maar zeggen niets over hoe al eerder genoemde interne structuren geïmplementeerd moeten worden. Concrete agent-architecturen gaan hier wel op in, ze geven een blueprint van de implementatie van de interne toestanden en de functies die samen zorgen voor de interne structuren. Deze structuren zorgen op hun beurt in de agent voor de perceptie, voor het proces van beslissingen nemen en de acties. Concrete architecturen worden verdeeld in vier typen, elk met een eigen kijk op de implementaties van de interne structuren van een agent.

9.2 Logic-based/deliberative architectures (op logica gebaseerde architecturen)

Dit type architecturen worden ook wel deliberative architectures genoemd. Ze zijn gebaseerd op de zogenaamde GOFAI: 'Good Old-Fashioned Artificial Intelligence'. De originele benadering op het bereiken van kunstmatige intelligentie. Deze benadering is gebaseerd op het bereiken van intelligentie door logica en probleem oplossen met manipulatie van symbolen, een idee dat samengevat is in de 'physical symbol systems hypothesis' *(Zie: Deel I - Ontstaan van de wetenschap*

10 Hoofdstuk 9 is onder andere gebaseerd op [5], [9], [10].

Kunstmatige Intelligentie- 2e pagina). Dit idee wordt toegepast bij de perceptie van de omgevingstoestand. De omgevingstoestand wordt omgezet naar een interne toestand die gerepresenteerd wordt in symbolen vorm. Zo ontstaat er een kennisrepresentatie in de agent die gebaseerd is op bijvoorbeeld eerste orde logica formules, plaatjes, natuurlijke taal zinnen, bayesiaanse netwerken of neurale netwerken. Hiernaast bezit zo'n agent een op logica gebaseerde a priori (ingebouwde) algemene theorie, ϕ, die het optimale gedrag van een agent beschrijft in de omgeving en die kennis over (het gedrag van) de omgeving bezit. Zo'n theorie bestaat doorgaans uit een set van regels. De agent maakt gebruik van de interne toestand en ϕ, en kan zo door middel van deductieve logica, plannen welke acties genomen moeten worden. Op deze manier is een agent volledig autonoom.

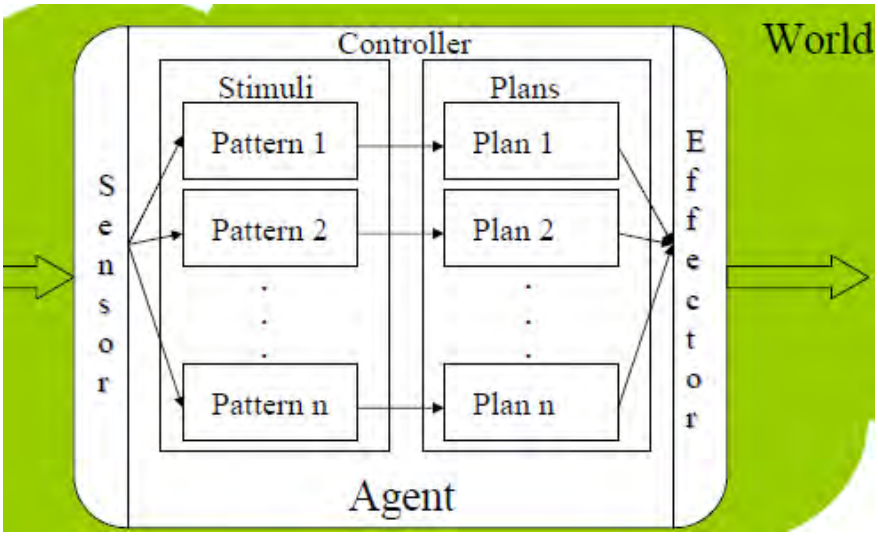


Figuur 14: Een representatie van de op logica gebaseerde architectuur [23]. Met links: de kennisrepresentatie: Het *Environment model* de interne toestand en *Domain knowledge* de ϕ. En rechts: de interpretatie hiervan en de bijbehorende acties gebaseerd op o.a. de doelen en de belangrijkheid van de doelen (*utility function*).

Deze type architecturen worden vooral gebruikt in gedeeltelijk waarneembare omgevingen vanwege de mogelijkheid van het modelleren en construeren (dmv. deductie) van het plannen van acties voor de gehele omgeving. Hier tegenover staat dat de op logica gebaseerde architecturen niet in een al te dynamische omgeving kunnen werken, omdat de complexiteit van symbool manipulatie ervoor zorgt dat de algoritmes zeer tijdsintensief zijn. Een andere nadeel van deze complexiteit is dat er bij het bouwen van zo'n systeem veel wiskundige kennis nodig is: de complexe wiskundige theorie ϕ moet ontwikkeld worden en up-to-date worden gehouden. Verder is het erg lastig om de omgeving, in een korte tijd, om te zetten naar een symbool beschrijving van de omgeving.

9.3 Reactive architectures (reagerende architecturen / op gedrag gebaseerde architecturen)

Het concept gedrag van een agent specificeert de manier waarop een agent reageert, welke acties het neemt, op de input van de sensoren vanuit de omgeving. Op gedrag gebaseerde architecturen gaan uit van dit concept, en coderen dit, a priori, in een directe indeling (mapping) van situatie (de omgevingstoestand) naar actie, bijvoorbeeld door middel van conditionele (if-then) regels. Het nadeel van dit idee is dat het aan de ontwikkelaar is om het gehele gedrag dat mogelijk zou kunnen zijn te coderen. Ook zal deze de agent direct geüpdate moeten worden, waarschijnlijk op veel plekken in de code, als er een kleine verandering is in de omgeving. Aan de andere kant, deze agenten zijn wel deterministisch en dus volledig te voorspellen, ze reageren snel op input en het is simpel om ze te ontwikkelen en ze te laten draaien op een infrastructuur.



Figuur 15: Een representatie van de Reactive Architecture [23]. Met links: de huidige omgevingstoestand wordt ingedeeld bij hiergenoemde *patterns*. Een pattern wordt gekoppeld aan rechts: een plan, wat dus de bijbehorende actie is.

Omdat ze zo simpel zijn, zijn reagerende architecturen alleen geschikt voor statische, volledig waarneembare omgevingen.

De intelligentie van deze agenten komt van het geheel van interactie van de agent met de omgeving en interactie tussen meerdere simpele gedragingen (acties).

9.4 Hybride architecturen

Omdat de twee genoemde concrete architecturen niet echt geschikt zijn voor dynamische omgevingen, zijn de hybride architecturen ontwikkeld. Door het op een bepaalde manier combineren van lagen logic based en lagen reactive architecturen worden de tekortkomingen opgevangen.

9.5 Belief-desire-intention (BDI) architectures

De meest populaire architectuur in de literatuur bestaat uit twee processen. Deliberation (beraadslaging), welke doelen moet de agent nastreven en means-end, hoe behaalt de agent deze doelen. Deze twee processen vallen onder praktisch redeneren[11], het gebruik van de rede om te beslissen hoe te handelen.

Het maken van beslissingen is in deze agenten afhankelijk van manipulatie van data structuren die de *beliefs* (overtuigingen), *desires* (verlangens) en *intentions* (bedoelingen) representeren. Door gebruik te maken van deze bdi concepten wordt het selecteren van een *plan* (reeks van acties, zie de opsomming hieronder) gescheiden van het uitvoeren van andere nog lopende plannen. BDI agenten kunnen dus een balans vinden tussen de tijd die ze bezig zijn met beraadslaging over plannen (wat te doen) en het uitvoeren van de plannen. Het is echter wel zo dat het ontwikkelen van de plannen buiten de scoop van het model ligt. De plannen worden vaak geselecteerd uit een plannen bibliotheek of komen van een externe planner applicatie. Dit maakt het implementeren van een leermechanisme in deze systemen lastig. Een ander nadeel van deze architectuur is dat deze niet ver

11 Theoretisch redeneren is het gebruik van de rede om te beslissen wat te geloven. Bijvoorbeeld: “agents use practical reason to decide whether to build a telescope, but theoretical reason to decide which of two theories of light and optics is the best.” [32]

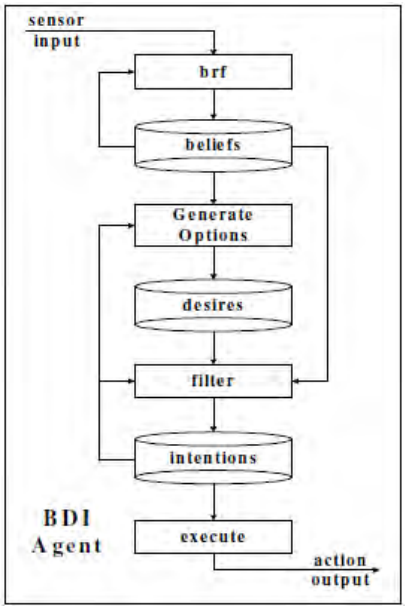
vooruit kan kijken. De beslissingen worden genomen op de gegevens (beliefs) van het moment en niet op die van de toekomst.

De belangrijkst componenten van de BDI architectuur:

1) **Belief**
Een verwachting (hoeft dus niet waar te zijn) van de agent over de huidige omgevingstoestand en over zichzelf. Omdat beliefs ook inferentieregels[12] kunnen bevatten, kunnen nieuwe beliefs afgeleid worden van de oude beliefs door middel van deductie. Werkt informerende voor een agent.

2) **Desire**
Dit komt neer op wat de agent graag zou willen behalen en werkt motiverend voor een agent. Voorbeelden van desires zijn: Vind de beste prijs, word rijk, ga naar het feest.
- **Doel**
  Een doel is een desire die door de agent is gekozen om te *proberen* te behalen. Om te voorkomen dat concurrerende desires (blijf thuis, ga naar het feest) doelen worden, moet de set van actieve desires al consistent zijn (dus geen concurrende desires).

3) **Intentions**
Intentions zijn desires waaraan een agent voor een groot gedeelte al vast zit om te behalen. De agent is begonnen met het uitvoeren van een *plan*, die ervoor zorgt dat een of meer intentions behaald worden.
- **Plannen**
  Plannen zijn opeenvolgingen van acties die de agent kan uitvoeren om de intentions te behalen. Plannen bestaan vaak uit andere plannen die gaande weg het proces ook geactiveerd worden. Zo bevat het plan om een stukje te gaan autorijden ook (eerst) het plan om de autosleutels te vinden. Het ontwikkelen van deze plannen hoort niet bij het BDI model.



Figuur 16: De BDI procedure schematisch weergegeven. [9]

12 Een voorbeeld van een inferentieregel: *Als* X een kikker is – *Dan* is X groen

De procedure van de BDI architectuur is te zien in *Figuur 16*. De volledige werking is als volgt.

1) *BRF*: belief revisie functie. Neemt de perceptie van de omgeving en de huidige beliefs van de agent en stelt daaruit een set van nieuwe beliefs samen.

2) *Generate options*: stelt op basis van de huidige beliefs en intentions een serie options samen. Deze options zijn de nieuwe desires.

3) *Filter*: Is het deliberation proces, die selecteert enkele van de desires, die vervolgens de intentions worden (Deliberative Architecture).

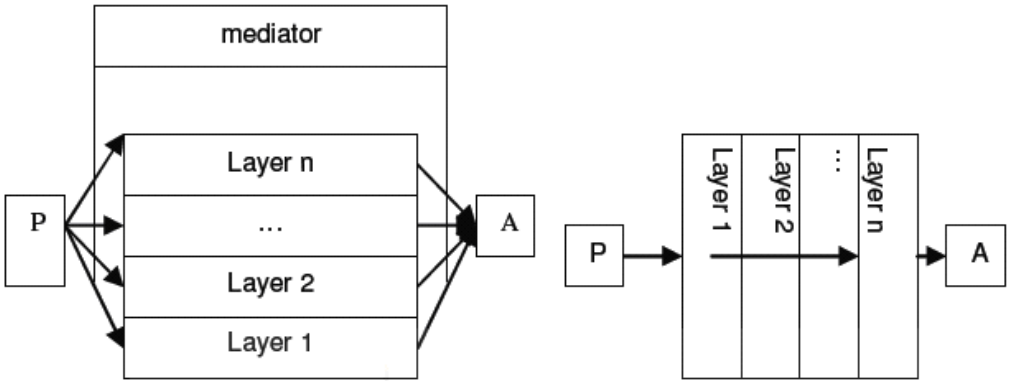
4) *Execute:* De intentions drijven de actie selectie functie, execute, die uiteindelijk de actie van de agent beslist (Reactive Architecture).

Een veelvoorkomend probleem in het ontwikkelen van agent-systemen is hoe de agent de 'aandacht' moet verdelen over beraadslaging over te nemen acties en uitvoeren van acties. Het BDI model zorgt voor een oplossing in het selecteren en uitvoeren van plannen.

Hoe elk individueel blok van de BDI precies werkt en hoe er een balans te vinden is tussen de twee processen deliberation en means-end zijn nog open vraagstukken, die voor elk project apart ingevuld moeten worden (of er wordt gekozen om te werken met een al bestaande architectuur van dit type).

9.6 Layered architectures (uit lagen bestaande architecturen)

In de layered architectuur hebben de reactive lagen een hogere prioriteit dan de deliberative. De layered architectuur bouwt modulaire agenten met variaties in reactive en delibarative mogelijkheden. Het ontwikkelen van de lagen kan op drie manieren gebeuren.

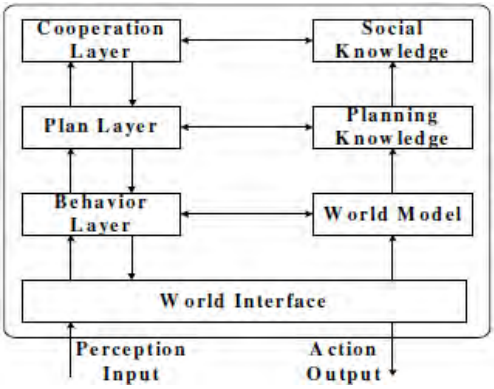


Figuur 17: Links de horizontale architectuur en rechts de 1-pad-verticale architectuur met P de Perceptie (input) en A de Actuatoren (output). [5]

De meest complexe hiervan is de horizontale architectuur: hierin is de input vanuit de omgeving direct verbonden met alle lagen, waardoor elke laag apart een beslissingen kan nemen voor een volgende actie. Omdat deze processen in de verschillende lagen gelijktijdig verlopen, is er een mediator laag nodig die eventuele conflicten over bepaalde beslissingen oplost.

De 1-pad-verticale architectuur: in tegenstelling tot de horizontale, bereikt in de verticale architectuur de input een voor een de lagen. De lagen geven de input door aan de volgende laag. Deze architectuur is zo opgebouwd dat alleen de laagste, of eerste, laag toegang heeft tot de originele input en alleen de hoogste laag heeft toegang tot de output, wat dus de actie is.

De 2-pad-verticale architectuur is in principe hetzelfde als de 1-pad, met het verschil dat alleen de laagste laag toegang heeft tot en de originele input en de output.



Figuur 18: Een 2-pad verticale architectuur voorbeeld. [9]

De verticale architecturen zorgen voor minder complexiteit in de interactie tussen de lagen dan de horizontale (geen mediator nodig) maar de opbouw van verticale architecturen zorgt er ook voor dat door een fout in een van de lagen het hele proces faalt.

In de interactie tussen de lagen een groot nadeel van layered architectures. In complexe agenten groeien de interacties tussen de lagen exponentieel met de groei van de lagen. Ook zorgt de opbouw van deze architecturen voor een ondoorzichtige structuur en is het dus moeilijk om het proces volledig te begrijpen.

Hoofdstuk 10 - Agent technologie[13]

10.1 Introductie

Tot nu toe is een overzicht gegeven van wat agenten zijn, hoe ze werken en hoe agenten ontwikkeld worden. Naast deze zaken is er ook een infrastructuur nodig voor de implementatie van agenten en de communicatie tussen agenten in een multi-agent-systeem. Agent technologieën zorgen voor deze infrastructuren.

10.2 Agent ontwikkel toolkits

Vanwege de groeiende populariteit van agent-systemen, komen er steeds meer *agent development toolkits* op de markt, die ondersteunen bij het bouwen van agent-systemen. De meeste van deze toolkits zijn gebaseerd op Java vanwege de platform onafhankelijke eigenschap. Met deze agent technologie kan er op een makkelijke manier agent-systemen gebouwd worden. Deze agent-systemen worden zo gebouwd dat ze data delen met andere applicaties binnen een bestaande organisatie of over het internet.

Aantal toolkits gebaseerd op Java:

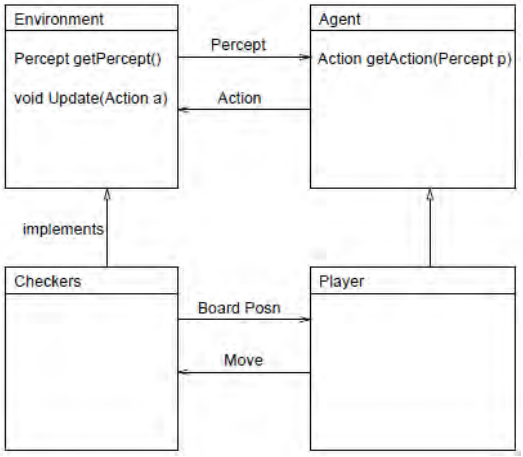
- Zeus
- Java Intelligent Agent Componentware (JIAC)
- SPADE *voor Multi-agent-systemen*
- IBM Agent Building Environment *Ook voor C++*
- JADE - Java Agent Development Framework

13 Hoofdstuk 10 is onder andere gebaseerd op [5], [8], [15].

- JACK Intelligent Agents *voornamelijk voor Multi-agent-systemen*
- Aglet Software Development Kit – ASDK *voor mobiele agent-systemen*

Een website met een tutorial over het agent-programmeren in JADE is bijvoorbeeld: http://www.iro.umontreal.ca/~vaucher/Agents/Jade/primer2.html

```
import jade.core.Agent;
import jade.core.behaviours.*;

public class myAgent extends Agent
{
    protected void setup()
    {
        addBehaviour( new myBehaviour( this ) );
    }


    class myBehaviour extends SimpleBehaviour
    {
        public myBehaviour(Agent a) {
            super(a);
        }

        public void action()
        {
...this is where the real programming goes
        }

        private boolean finished = false;

        public boolean done() {
            return finished;
        }

    } // ---------- End myBehaviour

}//end class myAgent
```



Figuur 19: Het gebruik van Java om agenten te programmeren.

10.3 Microsofts .Net framework

Dit is een framework voor het bouwen van applicaties die op Windows machines kunnen draaien. Microsofts .Net framework is gericht op de samenwerking van applicaties en bibliotheken die geschreven zijn in verschillende talen. Het blijkt zo te zijn dat de ingebouwde mogelijkheden van dit framework gebruikt kunnen worden voor het bouwen van agent-systemen.

Voor het bouwen van agent-systemen is de kracht van .Net dat verschillende componenten van een agent in verschillende talen geschreven kan worden (vooral handig bij extensies van het systeem). Het nadeel van het gebruik van .Net is dan weer dat agenten op Windows systemen moeten draaien en dit zorgt voor moeilijkheden bij 'interoperability'[14] (samenwerking met betrekking tot informatie) van agenten.

10.4 Agent communicatie

In 1996 werd de Foundation for intelligent Physical Agents kortweg FIPA opgericht. Een non-profit organisatie met als doel het bepalen van standaarden voor agenten met betrekking tot het bouwen van de software in de agenten en de communicatie van en tussen agenten. Ondanks dat een aantal van de toolkits de FIPA standaarden aanhouden, zijn ze niet verworden tot de standaard.

Een belangrijke standaardisatie van de FIPA is de standaard taal op het gebied van het ondersteunen van communicatie van informatie en interoperability tussen agenten. Waar voorheen de KQML taal werd gebruikt, is deze vervangen door de FIPA ACL (Agent Communication Language, de FIPA

14 Het vermogen van twee of meer systemen om informatie uit te wisselen en om deze uitgewisselde informatie onderling te gebruiken

standaard taal voor agent communicatie). ACL focust zich op het uitwisselen van berichten, die agenten formuleren en verzenden naar elkaar. Deze standaard definieert precies hoe zo'n bericht gestructureerd is wat betreft de codering, de semantiek (betekenis van de inhoud van het bericht zelf) en de pragmatiek (het belang van context van de communicatie in relatie tot het bericht).

In Bijlage 4 volgt een overzicht van een aantal parameters van de FIPA ACL. Deze uitgebreide beschrijving van de structuur van deze standaard kan gevonden worden op de FIPA website: http://www.fipa.org/specs/fipa00061/SC00061G.pdf (dit is de directe link naar specificatie van de FIPA ACL). Andere standaarden zijn ook te vinden op deze website.

Hoofdstuk 11 - Gebruik van agent-systemen

11.1 Kritiek op agent-systemen[15]

Ondanks de grote mogelijkheden van agent-systemen, wordt er ook kritiek geleverd. Zo zouden agenten neerkomen op uitgebreide Objecten (OO) of Expert Systemen. Uit alles wat tot hier besproken is blijkt het tegendeel. Het is juist zo dat Objecten en Expert systemen gebruikt kunnen worden in agenten. Voor het ontwikkelen van agenten wordt bijvoorbeeld veel Java (OO) gebruikt.

Het verschil met Objecten zit hem er in dat Objecten de controle hebben over de eigen interne toestand, maar niet over het eigen gedrag. Het idee van OOP is dat Objecten de *public* instanties en methodes van een ander Object op elk moment kan aanroepen. De *public* methoden worden dan uitgevoerd zonder dat het Object daar controle over heeft. Bij agenten werkt het zo dat agent A een aanvraag voor een actie moet doen bij agent B, waarna agent B beslist of deze actie voordelig is voor agent B. En alleen dan zal de aanvraag voor actie uitgevoerd worden. Verder zijn de basis eigenschappen die agenten moeten bezitten, reactief, proactief, sociaal, geen eigenschappen die Objecten standaard bezitten. Tot slot heeft elke agent apart een thread (een stroom door het systeem), en dus is er in een multi-agent-systeem een multi thread situatie, waar bij Objecten het vaak zo is dat het hele systeem maar bestaat uit één thread.

In the jaren tachtig van de vorige eeuw waren Expert Systemen de leidende KI technologie, geschikt voor het oplossen van problemen of adviseren in een bepaalde op kennis gebaseerde omgeving. Expert Systemen redeneren vanuit de kennis die ze ingebouwd hebben gekregen, en leren of ze de redenatie goed was via de feedback die ze krijgen op de oplossing of op het advies. Ondanks dat ze op deze manier intelligent kunnen lijken, en dus op agenten lijken zijn er toch verschillen. Zo zijn Expert Systemen, in tegenstelling tot agenten, vaak niet verbonden met de omgeving maar werken ze via een tussenpersoon. Bijvoorbeeld: hetzelfde diagnostisch Expert Systeem kan in meerdere ziekenhuizen gebruikt worden, met als tussenpersoon een dokter. Hierdoor zijn deze systemen ook niet in staat tot reactief, proactief of sociaal gedrag.

11.2 Een uitgewerkt, ultiem agent-systeem

Om tot een volledig begrip te komen wat agent-systemen kunnen, wordt in deze sectie uiteengezet hoe een 'ultiem beeld' van agent-systemen eruit ziet. Dit gebeurt aan de hand van zogenaamde *onbemande voertuigen*, waarvan een voorbeeld al genoemd is in de inleiding[1]. Het ontwikkelen van onbemande voertuigen, door middel van agent-systemen, geeft vooral een uitstekend beeld van wat de agent-systemen in de niet zo verre toekomst moeten kunnen.

15 Sectie 11.1 is onder andere gebaseerd op [37].

Zoals inmiddels bekend door dit werkstuk, is autonomie een belangrijk begrip in de agent-systemen. Omdat de autonome onbemande leger voertuigen in oorlogsgebieden ingezet gaan worden, moeten ze autonoom (zelf denken en handelen) hun weg kunnen vinden van A naar B. Op het moment dat er problemen opdoemen, zoals dreigingen of obstakels moeten ze autonoom bepalen wat de beste oplossing (uit te voeren actie) is. Het grote voordeel van autonomie in deze situaties is snelheid van denken en handelen.

Het is natuurlijk niet alleen voor het leger mogelijk om autonome onbemande voertuigen te ontwikkelen. In het vervolg van deze sectie wordt er gekeken naar wat er allemaal nodig is om met een agent-systeem een geautomatiseerde taxi chauffeur, een taxi zonder 'menselijke chauffeur', te ontwikkelen. Er is voor een taxi en de omgeving waarin een taxi gesitueerd is gekozen omdat hierbij een makkelijker beeld (een alledaagse situatie) te vormen is dan van een leger voertuig in een oorlog.

Figuur 20: Een taxi met chauffeur binnenkort verleden tijd?

Voordat gekeken wordt naar het ontwikkelen van een geautomatiseerde taxi chauffeur, een *taxi agent-systeem*, is het belangrijk om het volgende op te merken. Er wordt gesproken van een 'ultiem beeld' van agent-systemen, omdat de huidige agent-systemen eigenlijk beperkt worden door de technologieën waarvan ze gebruik maken. De beperkingen liggen voornamelijk bij de sensoren en de interpretatie die volgt op sensor informatie (voor het gemak valt dit onder het begrip sensor). Zaken als 'gezichtsherkenning', 'spraak interpretatie', 'camera beeld interpretatie' et cetera zullen nog de nodige ontwikkeling moeten maken opdat een ultiem agent-systeem ontwikkeld kan worden. Ditzelfde kan gezegd worden bij de actuatoren, en de interne wiskunde modellen (*zie sectie 9.2*). Er kan al veel, maar om de laatste stap te zetten moeten ook deze nog verder ontwikkeld worden. Samengevat is het concept van agent-systemen klaar voor de toekomst, maar de 'randtechnologieën' moeten nog stappen zetten om het maximale uit agent-systemen te halen. De huidige agent-systemen kunnen dus nog beter en completer (meer autonoom en intelligenter) worden. Het volgende voorbeeld van een geautomatiseerde taxi chauffeur is daarom een op dit moment nog wat onrealistisch systeem, maar zit niet heel ver af van wat nu al mogelijk is.

Het ontwikkelen van het taxi agent-systeem begint met het vaststellen van de complexiteit van de omgeving. Dit gebeurt door middel van het identificeren van een aantal eigenschappen van de omgeving zoals beschreven is in hoofdstuk 5. Voor de opsomming van de taxi agent-omgeving zie *Figuur 3*. Uit de opsomming blijkt dat deze agent-omgeving zeer complex is, wat het agent-systeem zeer lastig te ontwikkelen maakt

	Taxi
Fully observable	no
Deterministic	no
Episodic	no
Static	no
Discrete	no
Single agent	no

Figuur 21: De omgeving van de taxi blijkt zeer complex te zijn.

Het volgende dat moet gebeuren is het verkrijgen van een overzicht van de taak omgeving van de agent. Hiermee wordt bedoeld alles wat de agent nodig heeft en moet kunnen: wat en hoe het input moet krijgen en hoe en waarmee het acties moet uitvoeren (hoe de output gerealiseerd wordt). Verder is in de taak omgeving nog van belang de doelen die de agent moet behalen en hoe de omgeving van de agent er precies uitziet (deze is in ieder geval zeer complex, maar hoe ziet deze er concreet uit). Om dit overzicht te verkrijgen is een kort en bondig handvat ontwikkeld: het PEAS schema (beschreven in hoofdstuk 7).

PEAS schema voor het taxi agent-systeem:

1) *Performance Measures: juiste bestemming, veilig, snel, comfortabel, rekening houden met de verkeersregels en andere verkeersdeelnemers, maximaliseren van de winst, minimaliseren van slijtage van de auto, …*

De gewenste performance measures van het taxi agent-systeem moeten gezocht worden in zaken die vooral met het belangrijkste doel van het systeem te maken hebben: Het transporteren van passagiers van A naar B. Net als in veel gevallen zijn er conflicterende performance measures (bijvoorbeeld: comfortabel vs. snel), waarin dus een balans gevonden moet worden.

2) *Environment: verschillende wegen (snelweg, bebouwde kom), het klimaat, ander verkeer, voetgangers, de klanten, rijrichting (links of rechts) . . .*

De precieze omgeving bepaalt hoe moeilijk het wordt om de agent te ontwikkelen. Als de omgeving beperkt wordt door zaken als ‘er valt nooit sneeuw’ (klimaat) en er wordt alleen rechts gereden (de taxi komt nooit in een gebied waar er links wordt gereden) zorgt dit uiteraard voor het vergemakkelijken van het ontwikkelen van het taxi agent-systeem. Het model van de wereld (interne toestand) wordt hierdoor simpeler en de agent hoeft minder acties te kunnen uitvoeren (en hoeft dus over minder zaken te redeneren).

3) *Actuators: stuur, gaspedaal, rempedaal, claxon, lichten, output scherm …*

De actuatoren waarmee de agent acties kan uitvoeren zijn vergelijkbaar met degenen die beschikbaar zijn voor een menselijk chauffeur voor het ‘besturen’ van de taxi. Hiernaast heeft een taxi agent-systeem ook nog ‘iets’ nodig om output te communiceren naar de passagiers en misschien naar andere voertuigen. Dat ‘iets’ kan bijvoorbeeld een output scherm zijn, of veel geavanceerder, een ‘speech synthesiser’[16]. Merk in het geval van actuatoren als stuur, rempedaal, claxon, et cetera, op dat het wenselijk is om deze in de auto in te bouwen waar alleen het agent-systeem erbij kan en niet de passagiers (of autodieven). Een actuator zoals rempedaal moet dan ook niet al te letterlijk worden genomen, maar gezien worden als ‘een actuator die ervoor zorgt dat de auto kan remmen’.

[16] “Speech synthesis is the artificial production of human speech. A computer system used for this purpose is called a speech synthesizer”

Figuur 22: Enkele actuatoren (A) en sensoren (S) van het taxi agent-systeem in de taxi.

4) *Sensors: camera's, sonar, snelheidsmeter, GPS, microfoon, invoer voor de eindbestemming. .*

Waarschijnlijk het belangrijkste voor de agent is de input die het binnen kan krijgen. Voor deze input moeten sensoren zorgen. Via sensoren moet het taxi agent-systeem informatie krijgen over waar die is op dat moment, welke verkeersdeelnemers nog meer op de weg zijn, welke verkeersborden gelden (lezen van verkeersborden) en hoe snel de taxi en de andere verkeersdeelnemers gaan. Sensoren die hiervoor kunnen zorgen zijn camera's, snelheidsmeter en de kilometer teller. Verder moet de staat van de taxi gemeten kunnen worden door sensoren die de motor en de elektrische systemen meten en moet het systeem sensoren bevatten die ondersteuning bieden aan de controle en besturing van de taxi (bijvoorbeeld een versnellingsmeter voor op hellingen en in bochten). Denk hierbij aan systemen zoals die al gebruikt worden in de Formule-1, waarbij tijdens de race afstellingen van de auto kunnen worden aangepast. Voor precieze positie bepaling (op een kaart) kan het systeem gebruik maken van GPS. Infrarood sensoren of sonar sensoren zorgen voor informatie over afstanden tot obstakels en andere verkeersdeelnemers, wat bijvoorbeeld erg handig is bij het inparkeren. Ook moet het agent-systeem nog input krijgen van de passagiers, via een microphone (spraak) of toetsenbord, voor onder andere het opgeven van de bestemming en onderhandeling over de prijs.

Uit het PEAS schema blijkt wel dat zo'n taxi agent-systeem ontzettend veel sensoren en actuatoren nodig heeft. Als de huidige technologie, m.n. de sensoren (en de interpretatie hiervan) nog iets verbeteren is het ideaal beeld van een agent-systeem (in dit geval ingebouwd in een taxi) niet ver weg. Door het samenstellen van het schema, is nu het idee van het systeem, wat voor de ontwikkeling allemaal nodig is en waarmee het te maken kan krijgen, duidelijk gemaakt. Om nu de (interne) structuur van het taxi agent-systeem te maken, is er door de complexiteit van het systeem eigenlijk maar één keus voor het agent progamma (de connectie input – output). De eerste twee agent-programma's in hoofdstuk 8 (secties 8.1 en 8.2), gebaseerd op conditie-actie, vallen sowieso af. Immers, doordat de complexiteit van het taxi rijden zo groot is, de omgeving kan oneindig veel toestanden aannemen, is deze alleen te vangen in een oneindige **if-then** structuur, wat dus onmogelijk is. In het PEAS schema bij performance measures is eigenlijk al besproken waarom de 'op

doelen gebaseerde agent' (*sectie 8.3*) ook afvalt. Er zijn te veel mogelijke doelen, waardoor er wel conflicten tussen doelen moeten ontstaan. De utiliteit gebaseerde agent (*sectie 8.4*) zorgt voor een onderscheid in belangrijkheid (hoe blij wordt de agent van het doel) van te behalen doelen en is dus de juiste keus voor het taxi agent-systeem.

Vanaf hier wordt het ontwikkelen van de structuur van de agent zeer technisch, en kunnen er vele technieken worden gebruikt voor het opstellen van het omgevingsmodel (in *Figuur 12:* 'What the world is like now') en het beslissen welke actie volgt op de omgeving. Als vanuit de omgeving bijvoorbeeld input komt over de eindbestemming van net ingestapte passagiers, moet er daarop een wiskunde berekening plaatsvinden voor het vinden van een optimale route. Zo'n optimale route hoeft niet de snelste te zijn. De passagiers kunnen ook input geven in de zin van 'we willen niet door de stad' et cetera, ook hier zijn de mogelijkheden eindeloos. De wiskundige tools om dit soort zaken (search and planning) aan te pakken zijn voor handen. Vervolgens kan er misschien wel onderhandeld worden over de prijs (een interne structuur van de agent die onderhandelt), en wordt de prijs betaald door de passagiers.

De taxi gaat dan op weg, met behulp van onder andere de GPS, via de eerder gevonden optimale route. Voor de implementatie is het inzichtelijker om over te stappen op concrete architecturen (*hoofdstuk 9*). En dan met name de 'layered architectures', want aan de ene kant zijn er acties die bestaan uit simpele reactieve structuren (*sectie 9.3*): als er betaald is, start de motor. Aan de andere kant zijn er veel ingewikkeldere (logic based) structuren nodig. Voor het wegrijden van de auto moet eerst bepaald worden of het kan (via de camera's, sonars etc. wordt het omgevingsmodel geüpdate). Als het omgevingsmodel aangeeft dat de weg vrij is, wordt de actie wegrijden gestart. De beredenatie die nodig is om de actie te starten is op logica gebaseerd (*sectie 9.1*) en het daadwerkelijk gebruiken van de actuatoren (zet knipperlicht aan) kan dan gebaseerd zijn op een reactieve structuur. De 'output' die nodig is voor de actie komt neer op het actieveren van de actuatoren 'gaspedaal', 'linker knipperlicht aan' en 'stuur naar links' (mits de auto rechts staat geparkeerd). In *sectie 9.2* wordt het complete plaatje van de op logica gebaseerde acties uitgezet. De interne toestand en de ϕ bepalen hoe hard de taxi moet rijden, inhalen en anticiperen. Voor het bepalen van dit soort zaken zijn de in *9.2* besproken KI tools nodig zoals neurale netwerken, logica, bayesiaanse netwerken en alles wat je kan bedenken. Door de dynamische omgeving ligt op dit moment zeker een uitdaging in de ϕ, hoe wordt de algemene theorie ϕ gebruikt (*voor meer uitleg zie 9.2*).

Een duidelijk anticipatie voorbeeld is het volgende: in de omgeving, een stuk vooruit, staat iemand op de bushalte, en die geeft een signaal aan de bus die voor de taxi rijdt. De ϕ geeft aan als iemand op een bushalte staat én signaleert naar de bus is de kans groot dat de bus stopt. Het logische gevolg hiervan moet zijn dat de taxi van baan verandert als dit mogelijk is (indien de weg vrij is, worden dezelfde actuatoren geactiveerd zoals die bij het wegrijden. Dit soort anticipatie zorgt voor een veel betere verkeersdeelneming dan als de taxi 'wacht' totdat de bus aangeeft dat deze op de halte stopt (er hoeft niet op de rem te worden gestaan of op het laatste moment van baan gewisseld te worden om niet te botsen). Gedurende de rit komt de taxi allerlei situaties tegen waar deze mee moet omgaan (stoplichten, ander verkeer, omleidingen etc.). Als de taxi uiteindelijk zonder brokken op de bestemming arriveert, en een parkeerplek heeft gevonden, parkeert deze in. Het scant de omgeving via o.a. sonar sensoren, en door middel van het activeren van de juiste actuatoren ('versnelling in achteruit','knipperlicht aan', 'gaspedaal klein beetje in', 'sturen') wordt de taxi ingeparkeerd. Het inparkeren komt al veel voor in auto's ('park assistance'). Na het inparkeren, kan het taxi agent-systeem de passagiers nog een prettige dag, avond, of nacht wensen en kunnen de passagiers uitstappen.

Ook is het voor een agent-systeem (zoals deze) nog van belang om een leermechanisme te hebben. Als het agent-systeem fouten maakt, of niet geheel goede beslissingen neemt, zou het van grote intelligentie getuigen als het de volgende keer de zaken anders aanpakt. Sowieso scheelt een leermechanisme enorm veel programmeerwerk omdat niet elke (uitzonderings-) situatie uitgediept hoeft te worden. De agent kan simpel gezegd leren hoe die hiermee optimaal moet omgaan. Een voorstel voor het inbouwen van zo'n leermechanisme in een agent is besproken in *sectie 8.5* (*Figuur 13*). In de figuur is het 'performance element' het taxi agent-systeem zoals hierboven beschreven.

In een leermechanisme is feedback (in de figuur feedback van 'Critic') het belangrijkste onderdeel, omdat hierdoor bekend wordt of 'iets' goed is uitgevoerd of niet goed is uitgevoerd.

Het leermechanisme in het taxi agent-systeem werkt als volgt:

1) Stel, de taxi maakt op een gegeven moment een (te) scherpe bocht naar links, met een (te) hoge snelheid, dwars over een aantal rijbanen (gewenst gedrag is: in een eerder stadium naar links sorteren en dan 'rustig' linksaf slaan).
2) 'Critics' krijgt via de sensoren boze, vloekende passagiers en andere verkeersdeelnemers binnen. 'Critics' geeft als feedback (aan het 'learning element' dat de actie op deze manier niet hoort.
3) Het 'learning element' past dan de hier bijbehorende interne structuur van de agent (het 'performance element') aan naar: Op een rustigere manier linksaf slaan.
4) Voor het gebruik van de 'problem generator' valt te denken aan experimenteren met snelheden en remmen op verschillende soorten wegen (zandwegen, nieuw asfalt, slecht wegdek).

Figuur 23: Een boze klant geeft waardevolle feedback aan het taxi agent-systeem.

De input van de 'Critics' komt van bepaalde sensoren (invoer scherm, microfoon, camera). Er zijn tal van inputs te bedenken die 'leer' informatie geven. Zo kunnen de passagiers een cijfer voor de rit geven, welke is gebaseerd op de verwachting (snel, comfortabel, niet door de stad etc.) van het verloop van de rit. Andere inputs voor 'critics' zijn bijvoorbeeld de genoemde vloekende passagiers en bang of boos kijkende passagiers in reactie op bepaalde manoeuvres. Maar ook zaken als minder fooi en minder passagiers, de lengte van de remweg (voor de werking van de remmen) of de opbrengsten geven terugkoppel informatie aan het systeem. Het is duidelijk dat zo'n leermechanisme zeer waardevol is voor een intelligent agent-systeem.

De conclusie van het ultieme beeld van een intelligent agent-systeem dat hier is geschetst, is dat op de gebieden sensoren (en interpretatie hiervan), wiskundige representatie (ϕ), KI tools (neural nets, bayesiaanse netwerken) en de actuatoren nog enkele stappen gezet moeten worden voordat zo'n compleet systeem gerealiseerd kan worden. Dit beeld van intelligente agent-systemen is wel datgeen waar naartoe gewerkt moet worden. [19] beschrijft bijvoorbeeld een ideaal beeld van een agent-systeem in de medische wereld (ziekenhuis) die voor inkomende patiënten in samenwerking met de

staf: diagnosticeert, medische geschiedenis paraat heeft, bedden reserveert, operatie kamers reserveert, medewerkers voor de operatie verzamelt en nieuwe methodes voor operaties bestudeert (op het Web) en voorstelt aan degenen die moeten opereren. De huidige 'rand' technologieën beperken tot nu toe nog zo'n volledig agent-systeem als hierboven beschreven is, maar toch wordt het concept van de intelligente agent-systemen al op vele gebieden gebruikt. Wat volgt in het werkstuk in sectie 11.3 is dan ook een overzicht van het gebruik van intelligente agent-systemen.

11.3 Overzicht van agent-systemen[17]

Omdat in bijna elke situatie het idee van agent-systemen toegepast kan worden, is het onmogelijk om een volledig overzicht te geven van de soorten systemen. Daarom volgt hieronder een overzicht van enkele gebieden waar agent-systemen voorkomen, om een idee te krijgen waarin en waarvoor agent-systeem zoal gebruikt kunnen worden. In veel situaties is het zo dat er meerdere agenten samenwerken om een multi-agent-systeem te vormen. Meer hierover in deel III van het werkstuk.

11.3.1 Data mining agenten

Een agent met als doel het op een automatische, efficiënte manier vinden van informatie (patronen) in een *data warehouse.* Het probeert om relaties tussen delen van data te vinden, zodat het snel trends kan signaleren. Vasthoudendheid (het doel is gezet, en dat moet bereikt worden) en het gebruik van eerdere ervaringen maken agenten uitermate geschikt voor het doorzoeken van de gigantische hoeveelheden data in het data warehouse.

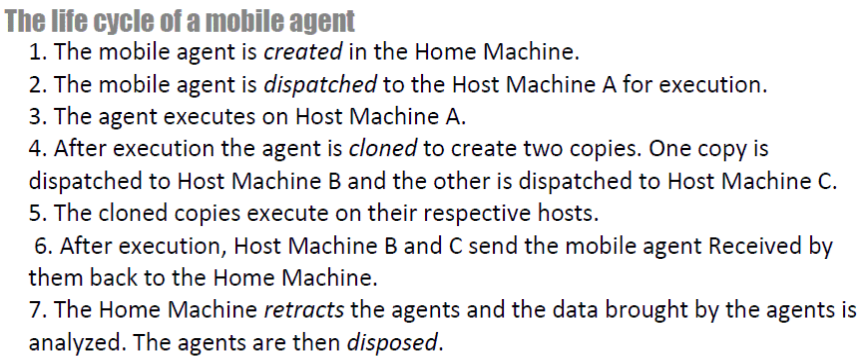
11.3.2 Intelligente autonome robots

Een vraag die opkomt bij het bestuderen van agent-systemen is in hoeverre deze systemen met robotica te maken hebben (alles omtrent ontwikkeling van robots). intelligente autonome robots voldoen zeker aan de eigenschappen van een agent-systeem, het interne van zo'n robot kan gezien worden als een agent-systeem (aangezien de robotica een eigen studiegebied is zijn er meerdere manieren om naar intelligente autonome robots te kijken). Het platform waarop de agent draait is dan de fysieke robot (het apparaat), met sensoren en actuatoren. Het bouwen van dit fysieke platform valt dan weer niet onder de studie van agent-systemen. Vooral in situaties van meerdere intelligente robots is het handig deze te beschouwen als multi-agent-systemen.

11.3.3 Mobiele agenten

Worden gebruikt in de zeer complexe wireless netwerken. Populaire toepassing hiervan is natuurlijk het WWAN (wireless wide area network**)**, waarin mobiele entiteiten zoals laptops, smartphone's en PDA's wireless met elkaar en met het internet worden. Het 'wireless internet' zorgt zo zelfs voor een globale connectie. De infrastructuur die nodig is voor wireless systemen moeten veilig en dynamisch (reken)ruimte toewijzen, users - hosts en terminals vinden in het netwerk, zorgen voor overdracht van informatie en zorgen voor coördinatie tussen verschillende soorten mobiele entiteiten. De mobiele agenten kunnen zich (de code, de huidige interne toestand en de data) over het wireless netwerk verplaatsen om de doelen te behalen. De beknopte werking (in het Engels) van een mobiele agent is te vinden in *Figuur 24*.

17 Sectie 11.2 is onder andere gebaseerd op [5], [11], [16], [22].



Figuur 24: Werking van een mobiele agent. Met Home Machine: de host waarop de agent geïnitieerd is. Dispatch: gestuurd. Cloned: exacte kopie gemaakt. Retracts: terugroepen. Disposed: vernietigd.

Mobiele agenten zorgen voor *minder belasting op het netwerk (betere bandwidth)*, want mobiele agenten doen de berekeningen etc. bij de hosts (in plaats van over het netwerk) en keren alleen terug naar de home machine met de resultaten. En omdat alleen de mobiele agenten worden verplaatst over het netwerk heeft het netwerk geen last van netwerk latency (vertragingen in het reizen van onder andere informatie van A naar B). Ook: in plaats van op elke host een protocol invoeren kan in de *agent dat protocol worden opgenomen*, zodat in het geval van veranderingen in het protocol alleen de agent geüpdate hoeft te worden. Verder zorgen mobiele agenten voor *asynchroon en autonome processen*, want zonder dat er connectie is met de home machine kan de agent processen uitvoeren. Een ander belangrijk aspect is dat deze agenten een *fout tolerantie* hebben. Als een host sluit, worden de agenten gewaarschuwd en verplaatsen ze zich naar een andere host om daar verder te gaan met het uitvoeren van processen. Een groot nadeel aan mobiliteit is de veiligheid. Doordat agenten zich continu verplaatsen is de kans groter dat ze bijvoorbeeld een virus op een host tegenkomen. Ook werkt het andersom, een virus kan zich voordoen als een mobiele agent en kan zo schade toebrengen aan de hosts.

Enkele populaire toepassingen van mobiele agenten komen aan bod in secties 11.3.3 en 11.3.4. Andere toepassingen zijn:

1) **Parallel berekeningen uitvoeren**
   Bij complexe problemen kunnen mobiele agenten ingezet worden om in een netwerk delen van het probleem op gespecialiseerde computers uit te voeren.

2) **Collectie van Data**
   Zoals eerder genoemd kunnen agenten naar hosts gestuurd worden, daar de berekeningen uitvoeren en dan de resultaten te sturen naar de home machine. Dit is uiteraard veel efficiënter dan alle data naar de home machine te sturen.

3) **Beperken van kosten van Wireless internet**
   Omdat er vaak kosten zitten aan internet connectie met een mobiele entiteit, kunnen agenten worden gestart en uitgevoerd, en kunnen de resultaten van het proces later worden opgehaald. Er is geen continue internet connectie nodig en het is goed voor het mileu aangezien de mobiele entiteit ook niet continu aan hoeft te staan.

11.3.3 Het web - search engines (zoekmachines)

Agent-systemen op het web die repetitieve taken van mensen overnemen worden vaak aangeduid met de term bot.

Web zoekmachines (Google, Altavista, Yahoo) gebruiken agent-systemen om automatisch het Web te doorzoeken, nieuwe websites aan de zoekindex van de zoekmachine toe te voegen, de informatie over de websites up-to-date te houden. Ook worden deze systemen gebruikt voor bijvoorbeeld het filteren van websites.

“Een spider (ook wel webcrawler genoemd) is een bot die het wereldwijde web op een methodische en geautomatiseerde manier doorbladert. Spiders maken veelal een lokale kopie van de gevonden pagina's om deze later te kunnen verwerken en indexeren voor bijvoorbeeld zoekmachines.

De werkingswijze is eenvoudig: de spider begint met een lijst met URL's en bezoekt deze één voor één, waarbij alle hyperlinks die in de bezochte pagina's voorkomen aan de lijst van te bezoeken URL's worden toegevoegd. Op deze wijze kan een spider vrijwel alle publiekelijk toegankelijke pagina's op het internet langsgaan. Veelal komen spiders met een vaste regelmaat langs om hun index actueel te houden.” [33]

11.3.4 Het web – E-commercie

In de vorm van shopping bots, gaan agenten het internet (of een ander netwerk) over om informatie te vinden over bepaalde items. Ze zijn er om klanten te assisteren bij het vergelijken van producten (prijzen), of om op unieke persoon gerichte advertenties te genereren, voor participatie in e-veilingen. Andere bekende bots zijn: Searchbots (voor het zoeken op het web zonder gebruik te maken van een zoekmachine), spambots (voor het verzamelen van e-mail adressen die op het internet rondslingeren), chatterbots (voor internet chat mens-agent, worden steeds meer ingezet naast een helpdesk).

11.3.5 Agenten in de financiële sector

Automatiseringen in de financiële sector betekent tijdsbesparing en dus geld besparing. Er zijn bijvoorbeeld *effectenhandelagenten* die verantwoordelijk zijn voor het managen van effecten. Zo’n agent zoekt bijvoorbeeld naar relevant nieuws en interpreteert de informatie over bepaalde effecten en managet de aan- en verkoop van effecten. *Investeer agenten* zijn agenten die aan de hand van bedrijfsinformatie beslissingen nemen over investeringen in bepaalde bedrijven.

In een ander deel van de financiële sector zijn er bijvoorbeeld *verzekeringsagenten* voor het creëren en onderhandelen van verzekeringsproducten voor een klant.

11.3.6 Agenten in de medische sector

Hierbij moet je bijvoorbeeld denken aan *medische assistentie agenten* die: Helpen bij het verzorgen van patiënten (eten, wassen, etc.), verantwoordelijk zijn voor informatie verstrekking van de dienstverleningen aan de patiënten en het monitoren en diagnosticeren van patiënten. Er zijn tevens specifiekere agenten zoals *gecoördineerde assistentie agenten*, voor hulp bij coördinatie van orgaan transplantaties tussen ziekencentra. Coördinatie bij het vrijkomen van een orgaan komt onder andere neer op het vinden van de meest geschikte ontvanger, het vinden van het beste transport en het samenstellen van het team en de bijbehorende apparatuur voor de operatie. Een ander voorbeeld zijn de *medische zorg agenten*, voor de zorg (o.a. monitoren, ordenen patiënt data, het versturen van herinneringen, versturen van hulp signalen aan het ziekencentrum) van ouderen thuis en in een ziekencentrum. Een uitgebreidere bespreking van agent-systemen, en een ultiem beeld hiervan, in de medische sector is ook te vinden in [19].

11.3.7 Menigte simulatie (zwerm intelligentie)

Een gebied dat steeds meer de aandacht krijgt is de simulatie van het gedrag van een grote kwantiteit entiteiten. Deze menigte simulatie is een toepassing van het idee van zwerm intelligentie (swarm intelligence)[18].

In de simulatie kunnen de entiteiten individueel gezien worden als een (vaak simpel) agent-systeem met mogelijkheden tot zien, horen, bewegen etc.: de entiteiten krijgen doelen en hebben interactie met andere entiteiten en de omgeving zoals een echte menigte dat zou kunnen. De mogelijkheid om zich aan te passen op veranderingen in de omgeving moeten ervoor zorgen dat de entiteiten allerlei (autonoom) gedrag vertonen, zoals het beklimmen van ladders, rennen, vechten, springen.

Gebieden waarin dit type simulatie gebruikt wordt:

- Films: Simulatie van veldslagen met veel karakters.
- Veiligheid: Trainen van emergency response teams (politie, brandweer, het leger) op het gedrag van menigtes in geval van nood.
- Optimalisatie van bepaalde menigte stromen: In pretparken, aan boord gaan van vliegtuigen.

Van menigte simulatie zijn een aantal voorbeeldfilmpjes te vinden op www.youtube.com, zoek op: mass simulation of kijk op [11].

11.3.8 Agent-projecten

Daadwerkelijke projecten en toepassingen van agent-systemen zijn te vinden op:

- **http://www.media.mit.edu/research/groups/software-agents**
  MIT media lab - The Software Agents group
- **http://www.cs.cmu.edu/~softagents/projects.html**
  The Intelligent Software Agents Lab, Carnegie Mellon University's Robotics Institute

18 SI systems are typically made up of a population of simple agents or boids interacting locally with one another and with their environment. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents. [34]

Deel III – Multi-agent-systemen[19]

Aan de hand van Deel I en Deel II van het werkstuk is er een beeld gegeven van intelligente agent-systemen. In dit derde deel wordt er gekeken naar systemen waarin meerdere van deze agenten opereren, interactie hebben met de omgeving en vooral met elkaar. Uiteindelijk moet dit leiden tot een 'samenwerking' waar alle in de omgeving opererende agent-systemen voordeel van hebben en waar het geheel voordeel van heeft.

Hoofdstuk 12 - MAS

12.1 Introductie

Er wordt van deze multi-agent-systemen (MAS)**,** ook wel genaamd op agenten gebaseerde systemen (agent based systems), gebruik gemaakt in situaties die meer vereisen dan één agent aankan. Zo'n single-agent is in zeer grote complexe systemen vaak te begrensd wat betreft kennis, rekenkracht en het perspectief (de input vanuit de omgeving). Ook zijn er situaties te bedenken waarin agenten (op bepaalde gebieden) tegengestelde belangen hebben. In een multi-agent-systeem zijn de agenten gekoppeld en kunnen ze in overleg een oplossing hiervoor vinden. Single-agenten zouden elkaar in zo'n situatie gaan tegenwerken (of zelfs saboteren) om zelf het optimale te krijgen. De kracht van MAS ligt hem dus in dat agenten individueel van elkaar werken en waar nodig samenwerken, of onderhandelen, op zo'n manier dat er synergie ontstaat (een MAS is beter dan de single-agenten bij elkaar opgeteld).

12.2 Karakteristieken

Omdat het al moeilijk is om een single-agent te definiëren, is het precies definiëren van een MAS nog moeilijker. Het idee van een MAS, zoals hierboven beschreven, is wel duidelijk en aan de hand hiervan zijn de karakteristieken van een multi-agent-systeem te beschrijven.

12.2.1 Heterogeen

Allereerst, kijkende vanuit de losse agenten die in het multi-agent-systeem draaien, wordt er gesproken over homogene en heterogene multi-agent-systemen. In veel multi-agent-systemen is het zo dat de agenten op verschillende manieren gebouwd zijn. De logische verklaring hiervoor is dat agenten achteraf worden toegevoegd op het moment dat dit nodig blijkt te zijn voor een betere werking van het systeem of dat de omgeving waarin de agenten opereren dusdanig veranderen dat de huidige agenten niet voldoende kunnen inspelen op het nieuwe deel van de omgeving. Heterogeniteit van een systeem kan ook gezien worden in de zin van dat de functionaliteiten van agenten niet hetzelfde zijn. Het blijkt dat er nogal onduidelijkheid bestaat over homogeniteit en heterogeniteit van multi-agent-systemen.

Een samenvattende omschrijving zou kunnen zijn dat de mate van heterogeniteit te maken heeft met in hoeverre de functionaliteit van alle agenten op elkaar lijken (werken ze op eenzelfde manier of op een totaal andere manier). Het is in ieder geval duidelijk dat bij het ontwikkelen van een multi-agent-systeem dit van belang is, hoe groter de heterogeniteit, hoe complexer het systeem is.

19 Deel III is onder andere gebaseerd op [10], [12], [18], [21], [24], [37].

12.2.2 Omgeving en de perceptie

De omgeving van een MAS is niet anders dan bij een single-agent. Het grote verschil zit hem in de individuele agenten. Deze hebben allen een gedeeltelijk waarneembare omgeving juist omdat ze zich alleen focussen op een deel van de omgeving. In een MAS gaat het erom de percepties van de agenten te combineren zodat ze meer kennis hebben over de omgeving dan alleen hun eigen perceptie en zo tot betere besluitvorming kunnen komen.

In situaties van zeer grote en complexe systemen kan er bijna geen single-agent ontwikkeld worden. Beter is het om de individuele agenten simpeler te houden zodat elke agent zich kan focussen op een bepaald onderdeel van het gehele systeem. Dit zorgt er wel voor dat agenten incomplete informatie en beperkte mogelijkheden hebben. Verder is het zo dat door de aanwezigheid van meerdere agenten logischerwijs de omgeving in een MAS dynamisch is.

12.2.3 Bestuur van een MAS

In tegenstelling tot veel andere systemen is er geen globaal bestuur in MAS. Elke agent bestuurt zichzelf, en het bestuur van het multi-agent-systeem als geheel komt vanuit de agenten en de samenwerking tussen de agenten.

Omdat er geen global bestuur is en de agenten elk voor zich een eigen onderdeel aanpakken, is er niet ergens in het system één grote data pool, maar is de info gedecentraliseerd over de agenten. Dit is juist een van de redenen waarom agenten in overleg moeten, de een weet meer over een bepaald onderwerp dan de ander.

12.2.4 Interactie

De kracht van MAS is dat de agenten losse entiteiten zijn en dus operaties van agenten tegelijk uitgevoerd kunnen worden. In situaties waarin agenten elkaar nodig hebben weerhoudt ze er niet van om tegelijk ook andere zaken te doen.

Hoewel in een MAS de agenten asynchroon van elkaar werken gebeurt het maken van beslissingen in een multi-agent-systeem op een samenwerkende manier. Het is juist hierdoor dat een MAS een goed resultaat kan behalen. De samenwerking die de agenten aangaan om tot beslissingen te komen kunnen worden beschreven met behulp van Game Theory. Om voor deze samenwerking te zorgen moeten de agenten met elkaar coördineren. Coordinatie zorgt ervoor dat individuele agent beslissingen resulteren in goede beslissingen voor het MAS (later meer hierover).

De interactie die agenten in een MAS met elkaar hebben moeten via een vorm van communicatie over bijvoorbeeld een netwerk gaan en het liefst in eenzelfde taal (zoals de FIPA ACL). Verder is een interactie (netwerk) protocol nodig voor een veilige, samenwerkende, tijdige, duidelijke uitwisseling van informatie. In een MAS is elke agent een zender en een ontvanger van berichten. Communicatie kan gaan over samenwerking van agenten, maar kan ook gebruikt worden voor onderhandelingen tussen agenten die alleen uit eigenbelang werken.

12.3 Mogelijkheden met MAS[20]

Een belangrijke kracht van een multi-agent-systeem ligt in situaties waarin meerdere gebieden samenkomen. Een voorbeeld hiervan is een distributie centrum. Hier worden verschillende goederen, op verschillende tijden door verschillende leveranciers geleverd en afgehaald. Bij het afleveren moeten de goederen onder andere opgeslagen worden. Ideaal zou het zijn als elk van deze

20 Sectie 12.3 is onder andere gebaseerd op [5], [21], [24].

gebieden eigen agenten hebben, en dat deze agenten samenwerken in één systeem: een multi-agent-systeem. In zo'n systeem worden levertijden, opslag tijden en afhaal tijden op elkaar afgestemd.

Een andere mogelijkheid die een MAS geeft is het blijven gebruiken van legacy systemen. Een agent kan zo gebouwd worden dat onderdeel van z'n uitvoer proces het aansturen van een legacy systeem is. Ook, omdat agenten losse entiteiten zijn in het MAS, is verspreide informatiebronnen geen enkel probleem. Verder is een Multi-agent-systeem robuust en betrouwbaar omdat het niet afhankelijk is van één systeem, bottleneck situaties in een agent geven geen problemen voor het hele systeem. En doordat er geen bottleneck situaties zijn, is het updaten of onderhoud geen probleem, want dit heeft vaak betrekking op één agent in plaats van op het hele systeem. Ook zijn de snelheid en efficiency van een multi-agente systeem hoog, omdat de agenten asynchroon draaien. En een MAS is flexibel en schaalbaar, omdat nieuwe agenten erg makkelijk toe te voegen zijn.

12.3.1 Domeinen

Voor een volledig begrip van deze sectie, bekijk ook *Deel II - Gebruik van agent-systemen*. Domeinen waarin MAS veel voorkomen zijn bijvoorbeeld:

1) **Het internet**
   Een open dynamische omgeving waarin agenten handelen namens gebruikers en met elkaar onderhandelen om toch de doelen te bereiken. Dit gebeurt veel in online veilingen,
   e-commerce, verspreidde data mining (de data bevindt zich op meerdere locaties in meerdere formaten, en de ene agent kan de andere 'vragen' om bepaalde data op te halen).

2) **Simulatie**
   Simulatie van sociaal gedrag in groepen waarin elk individu gemodelleerd wordt als een agent.

3) **Samenwerking in Robotica**
   Oftewel, meerdere robots (agenten) met eenzelfde doel. Een belangrijk, en bekend, testdomein hierin is het Robot voetbal. Waarin het multi-agent-systeem bijvoorbeeld gevormd wordt door twee robot teams die tegen elkaar voetbal spelen, met mensen als scheidsrechters (ook agenten). Elke robot heeft een andere functie, keeper – verdediger- middenvelder – aanvaller en in het team werken ze met elkaar tegen het andere team. De scheidsrechters zijn hier vooral opzichters maar maken wel onderdeel uit van het MAS: als er gefloten wordt moeten de robots stoppen.

4) **Planning**
   een groot gebied voor het gebruik van mas is de planning. Het eerder genoemde voorbeeld van het distributie centrum heeft alles te maken met planning. Hoe worden producten op het juiste moment geleverd. Een andere vorm van planning is te vinden in traffic control. Traffic control is het aansturen van verkeerslichten, opengooien van wegen, bepalen van omleidingen, opengooien van slagbomen en reguleren van snelheden, aan de hand van het verkeer (met name de verkeersdrukte). De agenten, die op verschillende locaties gestationeerd zijn en een beperkt gebied bestrijken, moeten de acties coördineren met de andere agenten,
   om zo bijvoorbeeld tot een optimale doorstroom van het verkeer te komen.

Hoofdstuk 13 - Coördinatie in MAS

13.1 Introductie

Omdat elke agent in een multi-agent-systeem een individuele entiteit is, met eigen perceptie – beslissingen nemen – actuatoren, worden de processen die door een MAS uitgevoerd worden gedecentraliseerde processen genoemd. Een MAS wordt daarom gezien als een verspreid systeem (distributed system). De interactie tussen agenten draait op het feit dat doelen van individuele agenten afhankelijk kunnen zijn van elkaar, agenten hebben elkaar dus nodig. Zo kan de een wel bij bepaalde input of output komen en de ander niet. Hebben twee agenten meer rekenkracht dan één agent. En kunnen twee agenten dezelfde resources nodig hebben, waardoor ze wel moeten onderhandelen. Het is duidelijk dat waar afhankelijkheden zich manifesteren agenten met elkaar coördineren moeten.

Belangrijk voor agenten in een MAS is dat ze extra sensoren en actuatoren hebben die gericht zijn op interactie met de andere agenten. Dit is nodig omdat er geen centraal bestuur is, en dus moet een agent zelf een andere agent kunnen onderscheiden in de omgeving, over de (acties van de) agent kunnen beredeneren en actie er op ondernemen mocht dit nodig zijn. Mogelijke acties zijn bijvoorbeeld manipulatie, overreden, argumentatie, overleg, bestuderen of commanderen.

Dat onderscheiden van en actie ondernemen op andere agenten gebeurt indirect door middel van manipulatie van gemeenschappelijke factoren van de omgeving. Of direct door middel van speciale sensoren/actuatoren die zorgen voor interactie via radio golven of via een netwerk, zoals het internet. Hierbij wordt een taal zoals de FIPA ACL gebruikt. In interactie tussen twee agenten wordt er onderscheid gemaakt tussen interactie in twee richtingen, *communicatie* genaamd en eenrichtingsinteractie, ook wel *observatie* genaamd. Bij dit laatste weet één van de agenten niet dat het geobserveerd wordt door de ander.

13.2 Soorten coördinatie

De afhankelijkheden die ervoor zorgen dat agenten moeten coördineren met elkaar, behoren niet allemaal tot dezelfde soort. Hierdoor zijn de manier van coördineren globaal in te delen in drie soorten.

13.2.1 Compete

Agenten kunnen wedijveren (*compete*) met elkaar over bepaalde resources. Agenten moeten dus onderling tot een akkoord komen die de eigen belangen maximaliseren. Ook wedijveren ze over het behalen van conflicterende doelen.

Problemen die bij deze concurrentie vooral ontstaan zijn dat agenten andere agenten proberen te beperken of zelfs beletten in het behalen van hun doelen. Zo verstrekken agenten foutieve informatie over de eigen identiteit, over andere agenten en over de omgeving. En proberen agenten beveiligde informatie te verkrijgen van en over andere agenten. Kortom ze liegen en bedriegen om maar zelf het eigen belang te maximaliseren.

Om deze problemen tegen te gaan wordt er vaak gewerkt met een doelfunctie, die gebaseerd is op de actie selecties van de agent, verwachte acties van andere agenten en de ontwikkelingen in de omgeving. Naast deze doelfunctie worden er in de MAS wetten, regels en procedures voor boetes geïmplementeerd, wat ervoor zorgt dat conflicterende agenten toch sociaal zijn ten opzichte van elkaar en zo ten opzichte van het gehele MAS waarin ze gesitueerd zijn.

13.2.2 Cooperate

Coöperatie (*cooperate*) van agenten komt neer op het behalen van congruente en complementaire doelen.

De problemen die aangepakt worden zijn vaak zeer complex en de informatie is verspreid, waardoor agenten elkaar nodig hebben om de eigen doelen te behalen.

13.2.3 Collaborate/negotiation

De derde vorm van coördinatie wordt ook wel samenwerking (*collaborate***)** of onderhandeling (*negotiation*) genoemd. Agenten werken samen om iets te bereiken en onderhandelen over gemeenschappelijke belangen.

Het overleg tussen agenten gaat meestal door middel van een aantal rondes waarin elke agent elke ronde een voorstel indient. Om dit te bewerkstelligen moet een agent een overleg set hebben bestaande uit voorstellen die de agent kan genereren, een protocol hebben die het overleg stuurt, een verborgen overleg strategie waarmee de agent de andere agenten moet overtreffen, een regel die aangeeft wanneer er consensus (een deal) is ontstaan over de onderhandelingen en wat de voorwaarden van de deal zijn.

13.3 Inbouwen van coördinatie

In de opsomming van de drie soorten coördinatie is er al wat gezegd over hoe de coördinaties tot stand komen in een MAS. Om dit te vervolledigen, wordt in dit deel nog in het kort besproken met welke ideeën, technieken of methodieken coördinatie ingebouwd kan worden in een MAS. Een uitstekend, uitgebreid boek hierover is: Shoham en Leyton-Brown [18]. Hierin worden allerlei methodieken besproken die betrekking hebben op de volgende twee soorten ideeën.

13.3.1 DPS

Zo is er *Distributed Problem Solving* (DPS, of Cooperative DPS) dat gebruikt wordt in multi-agent-systemen waarin agenten coöperatie hebben en samenwerken/onderhandelen om een gemeenschappelijk doel te bereiken. Problemen van grote orde kunnen met DPS gemakkelijk onderverdeeld worden in subproblemen, en deze problemen worden tegelijkertijd opgelost. In een MAS wordt een groot probleem gesplitst over meerdere agenten, met idealiter elk subprobleem toegewezen aan de agent die dat probleem het best aankan. Zo wordt heterogeniteit van de agenten in een MAS juist een kracht in plaats van een moeilijkheid en worden de verschillende informatie, computerkrachten van de agenten samengevoegd tot een synergie in het MAS.

Er zijn veel verschillende methodieken types die onder DPS vallen. Deze ideeën verschillen wat betreft de decompositie van het probleem en wat betreft het centraliseren van de oplossing (waar en op welke schaal moet de oplossing of de output plaatsvinden). Kaminka [12] noemt twee van deze typen.

In het eerste type ligt de focus van de agenten op verschillende subproblemen, maar ze hebben allemaal de beschikking over dezelfde input (al dan niet via een andere agent). De uitdaging hier ligt dus in de manier van decompositie van het probleem. In meer complexere situaties wordt de heterogeniteit in een MAS gebruikt om het juiste subprobleem aan de juiste agent uit te besteden. Een belangrijk objectief hierin is load-balancing, wat ervoor moet zorgen dat belasting van het probleem over alle agenten eerlijk verdeeld is. Een voorbeeld van dit type is *multi-agent computation*, verschillende delen van een complexe berekening (‘computation’) worden verdeeld

over verschillende agenten, en de resultaten worden gecombineerd op het moment dat alle agenten klaar zijn met hun proces.

Een andere type dat Kaminka [12] noemt is degene waarin het probleem door alle agenten op verschillende manieren (andere kennis, ervaringen, mogelijkheden) aangepakt wordt. De oplossing wordt gevonden door middel van een iteratief proces, waarin agenten komen met deelresultaten, die doorspelen naar de andere agenten ter assistentie en om verfijnd te worden, waarna de resultaten terug worden gestuurd. Uit deze iteraties van deelresultaten wordt er een globale oplossing geconstrueerd. Een voorbeeld van dit type is *distributed management of cellular Phone base-stations.* Hierin is een base-station een agent, die alleen over een beperkt gebied (een cel) met telefoons kan communiceren en deze kan monitoren (het telefoonverkeer kan regelen). Op gebieden waarin de cellen van stations elkaar overlappen, moet de frequentie en het resource gebruik (bandbreedte etc.) aangepast worden op de andere stations, zodat het verkeer geregeld kan worden door een ander base-station. Load-balancing zorgt er dan voor dat er niet een bepaalde cel is die te veel verkeer moet sturen.

13.3.2 Rationele en economische benaderingen

Het basis idee van DPS in MAS berust op agenten die zich verenigen om een gemeenschappelijk doel na te streven (of een probleem op te lossen). Maar eigenlijk zijn agenten meer individuele, rationele en zelfzuchtige entiteiten die de eigen doelen nastreven, die de eigen eerder besproken (zie 13.2.1) doelfunctie willen optimaliseren. Daarom is het een betere idee om verspreide, rationele benaderingen uit de economie en game-theory toe te passen die juist vanuit dit idee werken. Multi-agent-systemen waarin verschillende vakgebieden samenkomen bestaan meestal juist uit de uit eigen (vakgebied) belang werkende agenten. Belangrijk in deze systemen zijn de strategie, de reeks van acties, van de agenten en het protocol dat de interactie tussen de agenten regelt.

Ondanks dat de agenten vooral naar zichzelf kijken hoe goed ze werken, is er in het MAS toch een criteria voor hoe goed het totale systeem werkt. Dit kan heel simpel bepaald worden door middel van het *sociale welzijn*: de uitkomsten van de doelfuncties van de individuele agenten worden bij elkaar opgeteld. Een ander criteria voor de werking van een MAS is de stabiliteit van het systeem. *Stabiliteit* komt neer op dat agenten gemotiveerd zijn om, uit eigen belang, de gewenste strategieën te kiezen die uiteindelijk leiden tot een bepaald equilibrium (evenwicht) in het MAS.

Het ideale evenwicht houdt in dat er een optimaal compromis is tussen de agenten, waardoor deze niet meer de eigen strategie geheim hoeven te houden en ook geen moeite (tijd en resources moeten besteden aan) meer hoeven te doen om de strategieën van andere agenten uit te vogelen. Dan is er dus een *Nash equilibrium* (zie game-theory) ontstaan: geen van de agenten, gegeven de strategie van de anderen, kan een andere strategie kiezen die voor een beter eigen resultaat zorgt. Elke agent heeft dus een optimale strategie, ervan uitgaande dat de anderen hun strategie niet veranderen. Doordat alle agenten al een optimale strategie hebben, , hoeft geen van de agenten af te wijken van de huidige strategie, dus zijn alle strategieën in het evenwicht optimaal, want niemand kan zijn situatie verbeteren.

Een MAS is op z'n best als het het sociale welzijn maximaliseert én stabiel is. Echter zijn er situaties waarin deze twee criteria elkaar bijten, bijvoorbeeld in het geval van het *prisoner's dilemma* uit de game-theory (zie bijlage 5 voor uitgewerkte uitleg van het prisoner's dilemma). Hierin worden agenten voor de keuze gesteld om samen te werken of niet, waarin samenwerking leidt tot een beter resultaat wat betreft het sociale welzijn (in bijlage 5: opbrengst van de totale doelfunctie is dan 8), maar waarin de rationele keuze de agenten doet beslissen om niet samen te werken vanwege de stabiliteit (in bijlage 5: opbrengst van de totale doelfunctie is dan 4).

Nu de strategieën van de agenten ten opzichte van elkaar besproken is, rest alleen nog de protocollen voor de interactie. Kaminka [12] benoemt een aantal hoofd types, die voorkomen in vele verschillende variaties. Een type protocol gaat uit van *social choice* (stemmen); alle agenten geven een input om een bepaalde output te krijgen, en alle agenten zijn gebonden aan deze output. Een ander type is gebaseerd op *auctions* (veilingen), waarin alle agenten input geven, maar waarin de uitkomst alleen een deel van de agenten verbindt. Een van de types, *markets* (markten), is gebaseerd op de oude wetten in de economie, vraag en aanbod, die optimaliseren resource productie en consumptie door middel van overleg tussen consumenten en leveranciers/verkopers over de prijs. En tot slot zijn er de *contract nets*, wat vergelijkbaar is met de load-balancing in de DPS, die voor verspreide taak allocatie zorgen.

Hoofdstuk 14 - Voorbeeld cases MAS

Tot slot volgt nog een beperkte opsommingen van enkele voorbeelden en studies van multi-agent-systemen. Eerder zijn in Deel III al een aantal domeinen genoemd waarin multi-agent-systemen worden gebruikt.

Op de site van The Intelligent Software Agents Lab, zijn een aantal MAS te vinden:

- **http://www.cs.cmu.edu/~softagents/projects.html**

The Intelligent Software Agents Lab, Carnegie Mellon University's Robotics Institute

Een studie naar het gebruik van multi-agent-systemen in auto's:

- **http://www.cisa.informatics.ed.ac.uk/ssp/pubs/hering_msc.pdf**

cooperative multi-agent-systems in automobiles, Mirco Hering - 2004

Een studie naar het gebruik van multi-agent-systemen in het stockbroker wereldje:

- **http://opensiuc.lib.siu.edu/cgi/viewcontent.cgi?article=1019&context=cs_pubs&sei-redir=1#search=%22stockbroker+agent+system%22**

  TrAgent: A Multi-Agent-system for Stock Exchange, Tatikunta et al. - 2006

Een studie naar het gebruik van multi-agent-systemen in Intelligente Gebouwen:

- **http://www.softwareresearch.net/fileadmin/src/docs/teaching/WS05/SaI/Paper_Fuhrmann_Neuhofer.pdf**

Multi-Agent-systems for Environmental Control & Intelligent Buildings, Fuhrmann en Neuhofer-2006

Studies naar het gebruik van multi-agent-systemen in de Medische Sector:

- **http://www.openclinical.org/agents.html**

Allerlei voorbeelden

- ***http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.8155*** *(download)*

Medical Applications of Multi-Agent-systems, Antonio Moreno

Literatuurlijst

*Er wordt een hoofdletter in plaats van een jaartal gegeven als het jaartal niet bekend is.
**Wikipedia jaartallen zijn gebaseerd op de laatste modificatie

[1] AI Topics Editorial Board [Barrow Buchanan, Glick, Norvig et al.] (A),
http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/Agents,
Artikel: Agents - Intelligent Assistants Working With You and For You (MIT Encyclopedie)

[2] Bernhard Beckert (A)
http://www.uni-koblenz.de/~beckert/Lehre/KI-fuer-IM/03IntelligentAgents.pdf
Teaching slides (slide 31):

[3] B.G. Buchanan (A),
http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/BriefHistory

[4] Y. Demazeau (2000),
http://www-poleia.lip6.fr/~guessoum/asa/transparents/Demazeau.pdf
Slides

[5] M. Duma (2008),
Agents, Agent-architectures and multi-agent-systems
Master dissertatie

[6] Ulle Endriss (2006)
http://staff.science.uva.nl/~ulle/teaching/mas/slides/mas-game-theory-4up.pdf
Teaching slides

[7] S. Franklin (A)
http://www.msci.memphis.edu/~franklin/AgentProg.html
Persoonlijke website

[8] FIPA (A),
http://www.fipa.org/
FIPA website voor standaarden in agenten

[9] H. He, G. Chen and K. M. Goh (2005),
http://www.simtech.a-star.edu.sg/Research/TechnicalReports/TR04PR07.pdf,
Artikel: Constructing Agent-Based Systems

[10] Jane Hsu (A),
http://course.agent.csie.ntu.edu.tw/,
Teaching courses van Jane Hsu, informatieve slides.

[11] P. Jakubas (2010),
http://www.en.jakubas.net.pl/projects/multiagent-simulator ,
Website over menigte simulatie met behulp van agenten

[12] Gal A. Kaminka (2004),
Multi-Agent-systems,
Artikel in Encyclopedia of Human-Computer Interaction, Berkshire Publishing, 2004

[13] J. Linden (2001),
Iterative software engineering for multiagent-systems: the MASSIVE method,
Boek

[14] P. Maes (1991),
The agent network architecture (ANA),
Artikel in: SIGART Bulletin, 2(4):blz. 115-120.

[15] Microsoft (A),
http://www.microsoft.com/en-us/default.aspx ,
Microsoft website, search: .Net framework

[16] Manal Muneer (2009),
http://www.itswtech.org/Lec/Manal%28system%20programming%29/simeners_B/Mobile_Agent.pdf ,
Teaching course op University of technology-Baghdad, Irak.

[17] A. Orriols (2009),
http://www.albertorriols.net/?p=331,
Slides over geschiedenis van KI

[18] Y. Shoham en K. Leyton-Brown (2009),
MULTIAGENT-SYSTEMS, Algorithmic, Game-Theoretic, and Logical Foundations,
Boek

[19] Florian F. Schmitzberger (2006),
Agent-systems in the Medical Setting: The future of medical work
http://www.stanford.edu/~florians/files/MedicalAgents_Schmitzberger.pdf
Artikel

[20] Russell, Stuart J. en Norvig, Peter (2003),
Artificial Intelligence: A Modern Approach,
Boek

[21] Katia P. Sycara (1998),
Multiagent-systems,
Artikel in AI Magazine Volume 19 Number 2 (1998)

[22] R. Tatikunta, P. Shrestha, S. Rahimi, J. Bjursel (2006),
http://opensiuc.lib.siu.edu/cgi/viewcontent.cgi?article=1019&context=cs_pubs&sei-redir=1#search=%22intelligent+stockbroker+agents+program%22 ,
Artikel van southern Illinois University Carbondale

[23] Mihaela Ulieru (A)
http://www.uni-koblenz.de/~beckert/Lehre/KI-fuer-IM/03IntelligentAgents.pdf
teaching slides

[24] N. Vlassis (2003),
A Concise Introduction to Multiagent-systems and Distributed AI,
http://staff.science.uva.nl/~mmaris/class_2006_2007/cimasdai.pdf ,
Lecture Notes, UVA

[25] D.S. Weld, J. Marks, D.G. Bobrow (A), http://www.aaai.org/Library/Reports/nii.php#RTFToC55,
Artikel: The Role of Intelligent Systems in the National Information Infrastructure (MIT Encyclopedie)

[26] Wikipedia 1 (2008),
http://en.wikiversity.org/wiki/History_of_AI

[27] Wikipedia 2 (2011),
http://en.wikipedia.org/wiki/History_of_artificial_intelligence

[28] Wikipedia 3 (2011),
http://en.wikipedia.org/wiki/Logic_Theorist

[29] Wikipedia 4 (2011),
http://en.wikipedia.org/wiki/Physical_symbol_system

[30] Wikipedia 5 (2011),
http://nl.wikipedia.org/wiki/Principia_Mathematica

[31] Wikipedia 6 (2011),
http://en.wikipedia.org/wiki/Moravec%27s_paradox

[32] Wikipedia 7 (2011),
http://en.wikipedia.org/wiki/Practical_reason

[33] Wikipedia 8 (2011),
Wikipedia info: http://nl.wikipedia.org/wiki/Spider

[34] Wikipedia 9 (2011),
http://en.wikipedia.org/wiki/Swarm_intelligence

[35] Wikipedia 10 (2011),
http://nl.wikipedia.org/wiki/Turingtest

[36] Wikipedia 11 (2011),
http://en.wikipedia.org/wiki/Physical_symbol_system

[37] M. Wooldridge (2002),
An Introduction to Multiagent-systems,
Boek

[38] Wooldridge en Jennings (1995),
Intelligent agents: theory and practice,
Artikel in Knowledge Engineering Review (okt. 1994)

Bijlage 1

Bijlage 1.1: Turing test[21]

De Turingtest is een experiment, beschreven door Alan Turing in 1936, en nader uitgewerkt in zijn artikel Computing Machinery and Intelligence (1950) om licht te werpen op de vraag of een machine menselijke intelligentie kan vertonen.

Het artikel opent als volgt: "Ik stel voor om de vraag te beschouwen: kunnen machines denken? Dit moet beginnen met definities van de begrippen machine en denken." Dat is moeilijk, schrijft Turing. "In plaats van te proberen zo'n definitie te geven zal ik de vraag vervangen door een andere, die er nauw verwant mee is en uitgedrukt wordt in betrekkelijk eenduidige termen." Vervolgens stelt hij het Imitatiespel voor, dat sindsdien de Turingtest wordt genoemd:

Een menselijke ondervrager wordt in verbinding gesteld met een mens en met een computer en moet door ondervraging zien vast te stellen welke van de twee de mens is. De omstandigheden moeten zodanig zijn dat het om intelligentie gaat en niet om andere eigenschappen zoals bijvoorbeeld uiterlijke verschijning; daarom stelt Turing voor om de ondervraagden elders te plaatsen en als enige communicatievorm het uitwisselen van getypte tekst toe te staan, via "teletype"-machines, het tegenwoordige chatten.

Als de ondervrager niet consistent kan vertellen wie mens en wie machine is, doorstaat de machine de test. Dat is nog geen machine gelukt.

Het artikel is gewijd aan de vraag of zo'n machine inderdaad gemaakt zou kunnen worden, en wat de belangrijkste moeilijkheden zouden kunnen zijn. Turing concludeert dat hij geen enkele onoverkomelijke moeilijkheid ziet, behalve misschien paranormale gaven.

21 Bijlage 1.1 is gebaseerd op [35].

Bijlage 1.2: Physical symbol systems[22]

- **Formal logic**: the symbols are words like "and", "or", "not", "for all x" and so on. The expressions are statements in formal logic which can be true or false. The processes are the rules of logical deduction.
- **Algebra**: the symbols are "+", "×", "x", "y", "1", "2", "3", etc. The expressions are equations. The processes are the rules of algebra, that allow one to manipulate a mathematical expression and retain its truth.
- A **digital computer**: the symbols are zeros and ones of computer memory, the processes are the operations of the CPU that change memory.
- **Chess**: the symbols are the pieces, the processes are the legal chess moves, the expressions are the positions of all the pieces on the board.

The physical symbol system hypothesis claims that both of these are also examples of physical symbol systems:

- **Intelligent human thought**: the symbols are encoded in our brains. The expressions are thoughts. The processes are the mental operations of thinking.
- A running **artificial intelligence program**: The symbols are data. The expressions are more data. The processes are programs that manipulate the data.

22 Bijlage 1.2 is gebaseerd op [36].

Bijlage 2: Agent definities[23]

The MuBot Agent [http://www.crystaliz.com/logicware/mubot.html] *"The term agent is used to represent two orthogonal concepts. The first is the agent's ability for autonomous execution. The second is the agent's ability to perform domain oriented reasoning."*P> This pointer at definitions come from an online white paper by Sankar Virdhagriswaran of Crystaliz, Inc., defining mobile agent technology. Autonomous execution is clearly central to agency.

The AIMA Agent [Russell and Norvig 1995, page 33] *"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors."*

AIMA is an acronym for "Artificial Intelligence: a Modern Approach," a remarkably successful new AI text that was used in 200 colleges and universities in 1995. The authors were interested in software agents embodying AI techniques. Clearly, the AIMA definition depends heavily on what we take as the environment, and on what sensing and acting mean. If we define the environment as whatever provides input and receives output, and take receiving input to be sensing and producing output to be acting, every program is an agent. Thus, if we want to arrive at a useful contrast between agent and program, we must restrict at least some of the notions of environment, sensing and acting.

The Maes Agent [Maes 1995, page 108] *"Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed."*

Pattie Maes, of MIT's Media Lab, is one of the pioneers of agent research. She adds a crucial element to her definition of an agent: agents must act autonomously so as to "realize a set of goals." Also environments are restricted to being complex and dynamic. It's not clear whether this rules out a payroll program without further restrictions.

The KidSim Agent [Smith, Cypher and Spohrer 1994] *"Let us define an agent as a persistent software entity dedicated to a specific purpose. 'Persistent' distinguishes agents from subroutines; agents have their own ideas about how to accomplish tasks, their own agendas. 'Special purpose' distinguishes them from entire multifunction applications; agents are typically much smaller."*

The authors are with Apple. The explicit requirement of persistence is a new and important addition here. Though many agents are "special purpose" we suspect this is not an essential feature of agency.

The Hayes-Roth Agent [Hayes-Roth 1995] *Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions.*

Barbara Hayes-Roth of Stanford's Knowledge Systems Laboratory insists that agents reason during the process of action selection. If reasoning is interpreted broadly, her agent-architecture does allow for reflex actions as well as planned actions.

23 Bijlage 2 is gebaseerd op [7].

The IBM Agent [http://activist.gpl.ibm.com:81/WhitePaper/ptc2.htm] *"Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires."*

This definition, from IBM's Intelligent Agent Strategy white paper, views an intelligent agent as acting for another, with authority granted by the other. A typical example might be an information gathering agent, though the white paper talks of eight possible applications. Would you stretch "some degree of independence" to include a payroll program? What if it called itself on a certain day of the month?

The SodaBot Agent [Michael Coen http://www.ai.mit.edu/people/sodabot/slideshow/total/P001.html] *"Software agents are programs that engage in dialogs [and] negotiate and coordinate transfer of information."*

SodaBot is a development environment for software agent being constructed at the MIT AI Lab by Michael Coen. Note the apparently almost empty intersection between this definition and the preceding seven. we say "apparently" since negotiating, for example, requires both sensing and acting. And dialoging requires communication. Still the feeling of this definition is vastly different from the first few, and would seem to rule out almost all standard programs.

The Foner Agent [Lenny Foner - Download from ftp://media.mit.edu/pub/Foner/Papers/Julia/Agents--Julia.ps or online at http://foner.www.media.mit.edu/people/foner/Julia/ (click on "What's an agent? Crucial notions")]

Foner requires much more of an agent. His agents collaborate with their users to improve the accomplishment of the users' tasks. This requires, in addition to autonomy, that the agent dialog with the user, be trustworthy, and degrade gracefully in the face of a "communications mismatch." However, this quick paraphrase doesn't do justice to Foner's analysis.

The Brustoloni Agent [Brustoloni 1991, Franklin 1995, p. 265] *"Autonomous agents are systems capable of autonomous, purposeful action in the real world."*

The Brustoloni agent, unlike the prior agents, must live and act "in the real world." This definition excludes software agents and programs in general. Brustoloni also insists that his agents be "reactive &shyp; that is, be able to respond to external, asynchronous stimuli in a timely fashion."

As these definitions make clear, there's no general agreement as to what constitutes an agent, or as to how agents differ from programs. The Software Agents Mailing List on the Internet provides a FAQ (frequently asked questions) that says

Bijlage 3: Het Web - een complexe agent-omgeving[24]

Een tegenwoordig veelvoorkomend agent-omgeving is het World Wide Web. Vanwege de belangrijkheid van deze omgeving, zal in het kort de verschillende 'omgevingseigenschappen' van het WWW, aan de hand van de lijst hierboven worden aangegeven. Tevens is dit een mooi voorbeeld hoe omgevingen geduid dienen te worden.

Vanwege de complexiteit van het WWW zijn agenten uitermate geschikt om de kwesties die hierin voorkomen aan te pakken. Het is niet voor niks dat de komst van het WWW voor een boost heeft gezorgd in de ontwikkelingen op het gebied van agenten.

Opsommend is de WWW omgeving voor een agent:

1) **Gedeeltelijk waarneembaar**
   Omdat het Web zo groot is, is het onmogelijke voor een agent om alles waar te nemen. Ook is het zo dat een agent lang niet overal toegang heeft. Dit kan met geografie te maken hebben of met bepaalde rechten. Een agent bestrijkt dus altijd een deel van het Web, en zal zich dus moeten updaten indien informatie verplaatst wordt naar een plek buiten dit deel van het Web.

2) **Stochastisch**
   Op het Web zijn veel onzekerheden. Hierdoor zal de ene keer dat een agent een actie onderneemt een ander resultaat opleveren dan een andere keer als de agent dezelfde actie onderneemt. Zo wordt informatie veelvuldig verplaatst, zijn er veel schadelijke entiteiten (virussen,worms etc) en hebben servers te maken met downtime etc. Bij search engines (search agents) op het Web is er vaak onzekerheid over de input. De gebruikersinput kan meerdere betekenissen hebben, waarna de agent de verkeerde informatie voor de gebruiker ophaalt. Voor agenten is het van belang om inschattingen hierover te doen om toch een optimaal resultaat te behalen.

3) **Episodisch én sequentieel**
   Webbrowsers slaan informatie op in cookies en in de cache om toekomstig Webbrowsing te vergemakkelijk. Dit kan gezien worden als het sequentiële deel van het Web. Agenten kunnen dus prestaties uit het verleden gebruiken. De Webomgeving heeft ook episodische delen bijvoorbeeld in de vorm van captchas bij inloggen (bijv. elke keer een andere code om in te loggen).

4) **Dynamisch**
   Tegenwoordig worden de meeste websites dynamisch gecreëerd door middel van web programmeertalen. Hierdoor zijn de sites veiliger, efficiënter en bruikbaarder dan statische websites, die alleen gewijzigd kunnen worden als de auteur iets verandert. Het is duidelijk dat dynamische sites alleen wel zorgen voor grote complexiteit, waardoor agenten moeite hebben met het bijhouden van de huidige toestand van een website.

24 Bijlage 3 is onder andere gebaseerd op [5].

5) **Continu**
Op het Web komen constant nieuwe kwaadaardige entiteiten bij, (website)informatie wordt geüpdate, er komen nieuwe sites bij, sites verdwijnen, de technologie die het Web ondersteunen verbeteren en technologie waarop de agenten draaien verbeteren. Deze veranderingen die het Web continu ondergaat zijn zo veelvuldig, dat gesproken kan worden van een (bijna) oneindig grote toestandsruimte die de WWW omgeving kan aannemen.

6) **Single-agent en Multi-agent**
Voor sommige delen van het Web is een single-agent afdoende, voorbeelden hiervan zijn prijsvergelijkers. Andere meer complexere delen van het Web hebben meerdere, samenwerkende, agenten nodig om een doel te bereiken.

Bijlage 4: FIPA ACL parameters[25]

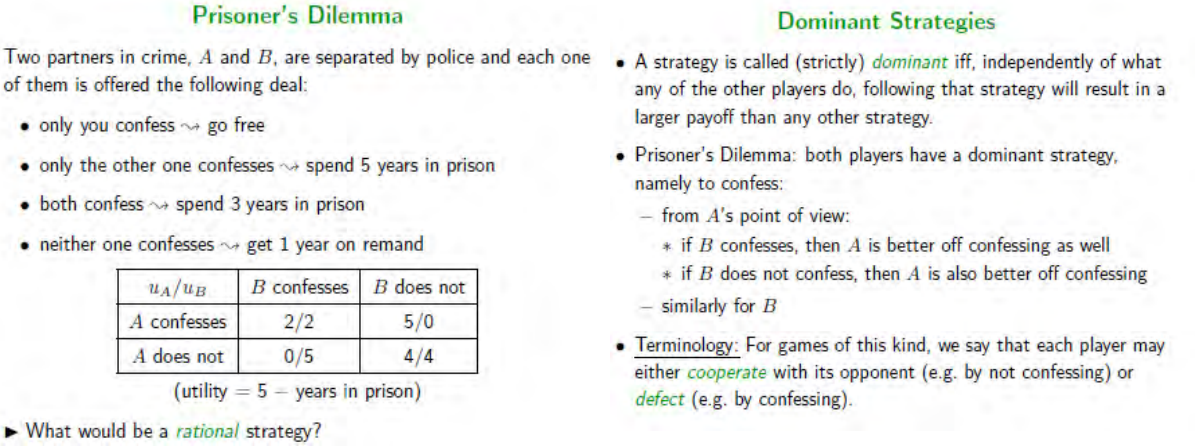
Waarbij de *Description* aangeeft waarvoor deze parameter dient en de *Parameter Category* de parameter indeelt in een bepaalde categorie.

Parameters	Description	Parameter Category
Performative	Mandatory for all ACL messages and it is used to indicate the ACL message kind of communicative act.	Type of communicative acts
Sender	Identity of the sender of the message.	Participants in communication
Receiver	The intended recipients of the message.	Participants in communication
Reply-to	Indicates that the message is to be directed to the agent mentioned in the reply-to parameter and not the sender agent.	Participants in communication
Content	Indicates the content of the message.	Content of message
Language	The language in which the content parameter is uttered.	Description of content
Encoding	Indicates the specific encoding of the content language expression.	Description of content
Ontology	Indicates the ontology(s) used to help interpret the symbols in the content expression.	Description of content
Protocol	Indicates the communication protocol that the sending agent is employing with this ACL message.	Control of conversation
Conversation-id	Conversation identifier that is used to help manage agent communication and activities.	Control of conversation
Reply-with	Expression that is used by the replying agent to identify this message.	Control of conversation
Reply-by	Indicates the latest time and/or date by which the agent that is sending would like to receive a response.	Control of conversation

25 Bijlage 4 is gebaseerd op [5].

Bijlage 5: Het prisoner's dilemma[26]

Zie *Figuur 25* voor de uitleg over hoe het prisoner's dilemma ontstaat.



Prisoner's Dilemma

Two partners in crime, A and B, are separated by police and each one of them is offered the following deal:

- only you confess $\rightsquigarrow$ go free
- only the other one confesses $\rightsquigarrow$ spend 5 years in prison
- both confess $\rightsquigarrow$ spend 3 years in prison
- neither one confesses $\rightsquigarrow$ get 1 year on remand

u_A/u_B	B confesses	B does not
A confesses	2/2	5/0
A does not	0/5	4/4

(utility $= 5 -$ years in prison)

▶ What would be a *rational* strategy?

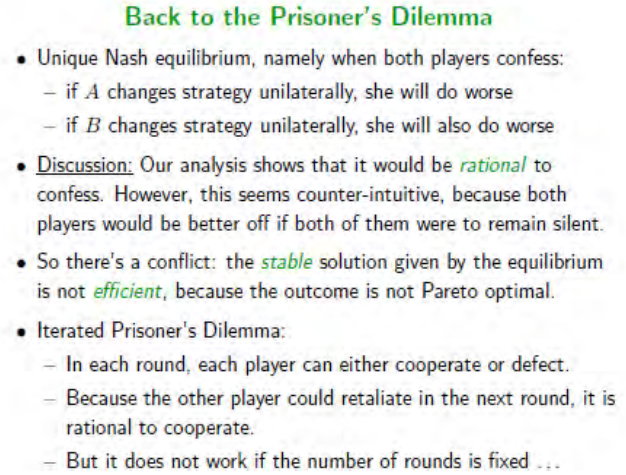
Dominant Strategies

- A strategy is called (strictly) *dominant* iff, independently of what any of the other players do, following that strategy will result in a larger payoff than any other strategy.
- Prisoner's Dilemma: both players have a dominant strategy, namely to confess:
  - from A's point of view:
    - if B confesses, then A is better off confessing as well
    - if B does not confess, then A is also better off confessing
  - similarly for B
- Terminology: For games of this kind, we say that each player may either *cooperate* with its opponent (e.g. by not confessing) or *defect* (e.g. by confessing).

Figuur 25: Links is de situatie geschetst, en rechts wordt beredeneerd wat de 'beste' strategie is voor deze situatie.

De doelfunctie hier (*Figuur 25*) is [5 – jaren in de gevangenis]. Het doel is om een zo hoog mogelijk resultaat te krijgen.

Individueel, rationeel beredeneerd wordt dit doel bereikt als de persoon bekent (ook wel de stabiele oplossing genoemd). Beide bekennen, en dan is voor beide het resultaat van de doelfunctie 2. Het totaal van de twee personen levert dan dus 4 op. Maar als de twee zouden samenwerken en beide niet bekennen, zou de totale doelfunctie hoger zijn (nl. 8) en zouden ze individueel gezien er ook beide beter van worden (beide 4).



Back to the Prisoner's Dilemma

- Unique Nash equilibrium, namely when both players confess:
  - if A changes strategy unilaterally, she will do worse
  - if B changes strategy unilaterally, she will also do worse
- Discussion: Our analysis shows that it would be *rational* to confess. However, this seems counter-intuitive, because both players would be better off if both of them were to remain silent.
- So there's a conflict: the *stable* solution given by the equilibrium is not *efficient*, because the outcome is not Pareto optimal.
- Iterated Prisoner's Dilemma:
  - In each round, each player can either cooperate or defect.
  - Because the other player could retaliate in the next round, it is rational to cooperate.
  - But it does not work if the number of rounds is fixed ...

Figuur 26: Het dilemma wordt hier nog eens samengevat.

De stabiele uitkomst zoals gegeven wordt door het equilibrium is niet efficiënt, want de uitkomst is niet Pareto optimaal. Een uitkomst van een spel is Pareto optimaal als er geen andere uitkomst is die ervoor zorgt dat elke speler evengoed af is en minstens een speler beter af is. Oftewel, een Pareto optimale uitkomst kan niet verbeterd worden zonder dat minstens een speler er slechter van wordt.

26 Bijlage 5 is gebaseerd op [6].