mei 2010

The Virtual Recipe Assistant

Hagenaars, R.

Faculteit Natuur en Techniek

Hogeschool Utrecht

Studentnummer: 1516548

Begeleiders: drs. L. van Moergestel (HU), dr. M. Dastani (UU)

Managementsamenvatting

Binnen de Agent Technology is er op het moment geen toonaangevende programmeertaal. Vrijwel alle agentgeörienteerde programmeertalen zijn momenteel nog in de ontwikkelfase. Zo ook 2APL, de agentprogrammeertaal van de Universiteit Utrecht. Om de concurrentiepositie van 2APL te bevorderen is het van belang dat de mogelijkheden hiervan getoond worden en de bruikbaarheid bewezen wordt door middel van demo's. Dit is de aanleiding geweest van de opdracht die ik bij de Universiteit Utrecht gedaan heb.

De doelstelling van dit project is als volgt gedefinieerd:

- Het ontwikkelen van een aansprekende demo voor 2APL, waarbij het model *Agent-Based Speech Act Generation* toegepast wordt en welke compatibel is met de *EIS standaard*.

Naar aanleiding van bovenstaande doelstelling heb ik de *Virtual Recipe Assistent* demo geschreven. Hierbij kan de gebruiker een virtueel recept koken en wordt deze hierbij begeleid door een 2APL-agent. Deze demo maakt gebruik van de eerder door mij opgeleverde *MSAgentEnvironment*. Dit deelproduct stelt een 2APL-environment in staat om *Microsoft Agents* aan te sturen, dit zijn grappige geanimeerde figuren.

De volledige beschrijving van de opgeleverde producten, de techniek hierachter en de uitvoering van het project zelf zijn in dit document beschreven.

Voorwoord

Tijdens enkele projecten, waaronder het bouwen van een Lego robot en een spel voor de Gameboy Advance, miste ik een bepaalde techniek in het programmeren van software. We waren in staat om een robot door de kamer te laten rijden zonder dat deze objecten raakte en in ons Gameboy spel liep een monstertje rond dat voortdurend jacht op de speler maakte. Deze programma's doorliepen simpelweg een reeks instructies en waren dus verre van intelligent. Hoezeer had het toepassen van kunstmatige intelligentie deze projecten interessanter kunnen maken!

Gedurende het eerste semester van het vierde jaar van mijn opleiding volgde ik samen met twee medestudenten een onderzoekssemester. We kregen de keuze uit een aantal onderwerpen waarin we ons konden verdiepen voor gedurende een half jaar. Al snel hadden we onze keuze voor het onderwerp Agent Technology gemaakt.
Tijdens dit onderzoek heb ik al kennis gemaakt met het fenomeen *agent* en *multiagent systems* en was mijn interesse voor meer diepgang in deze theorie gewekt.

Toen ik vernam dat een medestudent, Tim Theeuwes, bezig was met een afstudeerproject in de Agent Technology aan de Universiteit Utrecht heb ik besloten hetzelfde te willen doen. Aldus heeft Leo van Moergestel, mijn onderzoeksbegeleider destijds, contact opgenomen met Mehdi Dastani van de UU. Mehdi bleek geïnteresseerd te zijn in nog een student van de Hogeschool Utrecht en zodoende ben ik aan mijn afstudeerproject gekomen.

Er is een aantal personen die ik graag wil bedanken voor de geleverde bijdrage aan mijn project. Allereerst noem ik Leo van Moergestel, voor het immer getoonde enthousiasme en de altijd aanwezige bereidheid om te helpen en te ondersteunen indien mogelijk. Ik heb onze gesprekken altijd als zeer plezierig ervaren. Ook wil ik Mehdi Dastani bedanken, in de eerste plaats voor het aannemen van mij als afstudeerder, maar ook voor zijn betrokkenheid bij de ontwikkeling van het product en zijn colleges die ik bij mocht wonen. Daarnaast bedank ik Nieske Vergunst voor haar bijdrage in de vorm van het model dat het project nog interessanter heeft gemaakt. Tot slot bedank ik Tristan Behrens voor zijn hulp omtrent het “Eisify-en” van mijn environment en zijn geduld met mij.
En natuurlijk bedank ik vrienden en familie voor de nodige mentale steun en het willen nalezen van dit document, allen hartelijk dank.

Ramòn Hagenaars
Utrecht, mei 2010

Gebruikte symbolen

In dit document wordt een aantal symbolen gebruikt om de leesbaarheid te bevorderen en extra aandacht te vragen voor specifieke zaken. Hieronder bevindt zich een overzichtje van alle symbolen in dit document waaruit de betekenis per symbool afgelezen kan worden:

De volgende zin is een feit; *niet* gebaseerd op een keuze.

Er wordt een voorbeeld gegeven om iets te verduidelijken.

Er zal een keuze toegelicht worden.

Gebruikte logica

In enkele delen van dit document is *predicatenlogica* toegepast in combinatie met tekens van de *verzamelingsleer* om exacte betekenissen toe te schrijven aan bepaalde onderdelen of deze te verduidelijken.

In de predicatenlogica worden eigenschappen van en relaties tussen objecten formeel beschreven. Een predicaat zegt iets over een *subject* in de vorm $P(s)$, waarbij P de predicaat vormt die toegepast is op s. Met een predicaat kun je een eigenschap van een subject definiëren. We zouden bijvoorbeeld kunnen definiëren dat in het zojuist neergezette predicaat P voor Persoon staat en s voor Sjaak: Sjaak is dan kennelijk een persoon.

Hieronder staat een overzicht van de gebruikte symbolen met per symbool een korte beschrijving:

Symbol	**Beschrijving**
$\neg x$	x geldt niet
$x \wedge y$	x en y gelden
$x \vee y$	x of x geldt
$x \in y$	x is een element van y
$\exists x(y)$	er bestaat een x waarvoor geldt y
$\forall x(y)$	voor alle x geldt y
$x \rightarrow y$	wanneer x geldt dan geldt y. Wordt soms ook gebruikt als: x wordt y
$x \Rightarrow y$	wanneer x geldt dan en slechts dan geldt y

Inhoud

1 Inleiding

In het begin van de jaren 1990 kende het paradigma bekend als o*bjectgeoriënteerd programmeren* een opmars binnen de informatica. Deze geheel andere visie op de software ontwikkeling heeft zich een belangrijke plaats toegeëigend met talen als Ruby, Smalltalk, Eiffel en ook C++ en Java. Tegenwoordig is het *OO* paradigma niet meer weg te denken uit de software wereld.[a]
In 1993 introduceerde Yoav Shoham het zogeheten a*gentgeoriënteerd programmeren* (AOP) met de agent programmeertaal AGENT0 waarmee wellicht opnieuw een nieuwe fase binnen de software development ingeluid is.[b] Er zijn sindsdien meerdere agentgeörienteerde programmeertalen ontwikkeld waaronder Jason, Goal, Jadex en ook 2APL waar dit project op gebaseerd is.

Op het moment dat deze tekst geschreven wordt is er nog geen standaard agent-programmeertaal en zijn de bestaande agent programmeertalen nog volop in de ontwikkeling. Ook 2APL bevindt zich in de ontwikkelingsfase en de ontwerpers doen hard hun best om voor deze taal een gedegen plaats binnen de *Agent Technology* te veroveren. Hierbij is het van belang om te bewijzen dat de agenttaal daadwerkelijk in de praktijk gebruikt zou kunnen worden.

Door het ontwikkelen van een aansprekende demo die gebruik maakt van de mogelijkheden van 2APL hoopt de *Intelligent Systems Group* van de Universiteit Utrecht op een goede concurrentiepositie van 2APL. Aldus is het project *The Virtual Recipe Assistant* ontstaan.
De Virtual Recipe Assistant toont op een speelse manier aan hoe 2APL ingezet kan worden om leuke en interessante programma's te ontwikkelen met software agents. In deze demo kan de gebruiker een virtueel recept koken waarbij deze begeleid wordt door een *pratende* assistent naar keuze. Over de vraag of de Virtual Recipe Assistant aansprekend is, kan dus in ieder geval niet meer getwijfeld worden.

1.1 Doelstelling

In het plan van aanpak, dat opgeleverd is op 5 februari, is de doelstelling voor dit project gedefinieerd. Dit plan van aanpak is toegevoegd aan de bijlagen, zie hiervoor bijlage A. Deze doelstelling is als volgt gedefinieerd:

- Het ontwikkelen van een aansprekende demo voor 2APL, waarbij het model *Agent-Based Speech Act Generation*[1] toegepast wordt en welke compatibel is met de *EIS standaard*[2].

In dit document staat deze doelstelling centraal en wordt verdedigd waarom deze al dan niet als gehaald beschouwd zou kunnen worden. De conclusie zal dit samenvatten in een bondig antwoord op deze doelstelling.

1.2 Opgeleverde producten

Naast het eindproduct dat aan de doelstelling zou moeten voldoen, zijn er nog twee deelproducten opgeleverd. We onderscheiden de volgende opgeleverde producten:

- MSAgentEnvironment, voorzien van Javadoc;
- De Virtual Recipe;
- De Virtual Recipe Assistent, het eindproduct.

1 Zie 5.2.2 : Model Agent-Based Speech Act Generation
2 Zie 3.2 : Environment Interface Standard

1.3 Documentopbouw

Hoe is dit dit document opgebouwd? Los van alle document-gerelateerde inhoud wordt uitgegaan van vijf delen:

i. Het eerste deel (pagina 15 t/m 18) introduceert de lezer in de Agent Technologie en enkele andere achtergronden. Aangezien er verschillende vaktermen uit deze technologie gebruikt worden in de latere hoofdstukken, zullen deze hier vooraf uitgelegd worden;
ii. Het tweede deel (pagina 19 t/m 37) beschrijft het product en de deelproducten die opgeleverd zijn, hoe deze werken en waarom voor deze werking gekozen is;
iii. Vervolgens wordt in het derde deel (pagina 39 t/m 47) de techniek achter deze producten beschreven. Gebruikte technieken, technische keuzes alsmede de gebruike tools en alle aanverwante onderwerpen zijn hier terug te vinden;
iv. Het vierde deel (pagina 49 t/m 51) gaat over de totstandkoming van de opgeleverde producten. Hier wordt de projectmatige aanpak weergegeven;
v. Tot slot zal een conclusie worden weergegeven als antwoord op de in het plan van aanpak gestelde doelstelling (pagina 53).

1.4 Externe documenten

Achterin is een aantal externe documenten als bijlage toegevoegd, namelijk:

A) Het plan van aanpak behorende bij dit project;
B) EIS-documentation - MSAgentEnvironment;
C) Userguide to MSAgentEnvironment;
D) EIS-documentation - VirtualRecipeAssistentEnvironment;
E) Een beknopte inleiding tot JNI.

De eerste bijlage, Bijlage A: het plan van aanpak, wordt meerdere malen aangehaald in verband met de opdracht die hierin beschreven is. De bijlagen B en D zijn voornamelijk interessant voor de lezers die mogelijk gebruik willen maken van de MSAgentEnvironment en de Virtual Recipe Assistent, of deze willen doorontwikkelen. Bijlage C omschrijft de stappen die gedaan moeten worden om de MSAgentEnvironment (en dus indirect ook de Virtual Recipe Assistent) werkend te krijgen.

2 Achtergrond

Door dit gehele document wordt een aantal begrippen gebruikt die wellicht voor een lezer zonder inhoudelijke kennis hiervan tot verwarring kan leiden. Dit hoofdstuk geeft met name een korte inleiding in de terminologie van de Agent Technology waarop dit project gebaseerd is. Daarnaast zullen nog enkele andere termen toegelicht worden.

2.1 Agent Technology

De Agent Technology is de studie die zich bezighoudt met het definiëren, ontwerpen en toepassen van agenten in computer software. Het is een wetenschap die veel weg heeft van haar wat meer bekende zus: de *Artificial Intelligence (AI)*, ofwel de *Kunstmatige Intelligentie*, waarbij gestreefd wordt naar het ontwerpen van denkende machines.
Een verschil tussen AI en Agent Technology is dat AI dieper ingaat op de intelligentie van een individuele agent. De Agent Technology daarentegen is meer gericht op sociale aspecten; de capaciteit om samen te werken waarbij één of meerdere doelen gehaald worden.[b]

2.1.1 Wat is een agent?

Op dit moment is er nog geen universeel geaccepteerde definitie van de term agent. Ook de eigenschappen die men toekent aan een agent kunnen verschillen. Een voorbeeld dat *Michael Wooldridge*[3] noemt is *de eigenschap om te leren;* in sommige gevallen is een leerproces voor een agent van groot belang, terwijl in andere gevallen dit niet wenselijk is.

Om het onderwerp van Agent Technology bespreekbaar te maken zullen we in dit document uitgaan van de volgende definitie van een agent:

Een agent is een computersysteem dat geplaatst is in een *environment* en dat in staat is tot het uitvoeren van *autonome acties* binnen deze environment om zijn doelen te behalen. [Wooldridge and Jennings, 1995]

Bij implementatie van agenten in software zijn de agenten vrijwel altijd binnen een eigen thread of proces geplaatst om extra te benadrukken dat een agent een individuele entiteit is en dat de agent zijn eigen gang kan gaan. [b]

2.1.2 Wat is een environment?

In de definitie van een agent hierboven kwam het woord *environment* al naar voren. De letterlijke vertaling van het woord is *milieu*: het geheel van de natuurlijke, maatschappelijke en culturele omgeving dat op een levend wezen zijn invloed doet gelden [c]. Hoewel hier geen sprake is van levende wezens, maar van agents komt deze beschrijving redelijk in de buurt van wat verstaan wordt onder environment in de Agent Technology.
Een environment is een leefomgeving waarbinnen meerdere agenten in en uit kunnen lopen (afhankelijk van de toegankelijkheid van de environment) en waarbinnen de agenten gebruik kunnen maken van de aangeboden faciliteiten.

2.1.3 Multi Agent Systems

Een *Multi Agent System*, vaak afgekort met *MAS*, is een systeem waarbij *meerdere agenten* betrokken zijn. De verhoudingen tussen deze agenten kunnen verschillen van coöperatief tot competitief.
Een voorbeeld van een type MAS is de zogenaamde *Swarm Intelligence*. Hierbij werkt een groot aantal simpele agenten aan een gemeenschappelijk doel. Voorbeelden hiervan vinden we terug in de natuur in bijvoorbeeld: mierenhopen, scholen van vissen en bacteriële groei.[d]

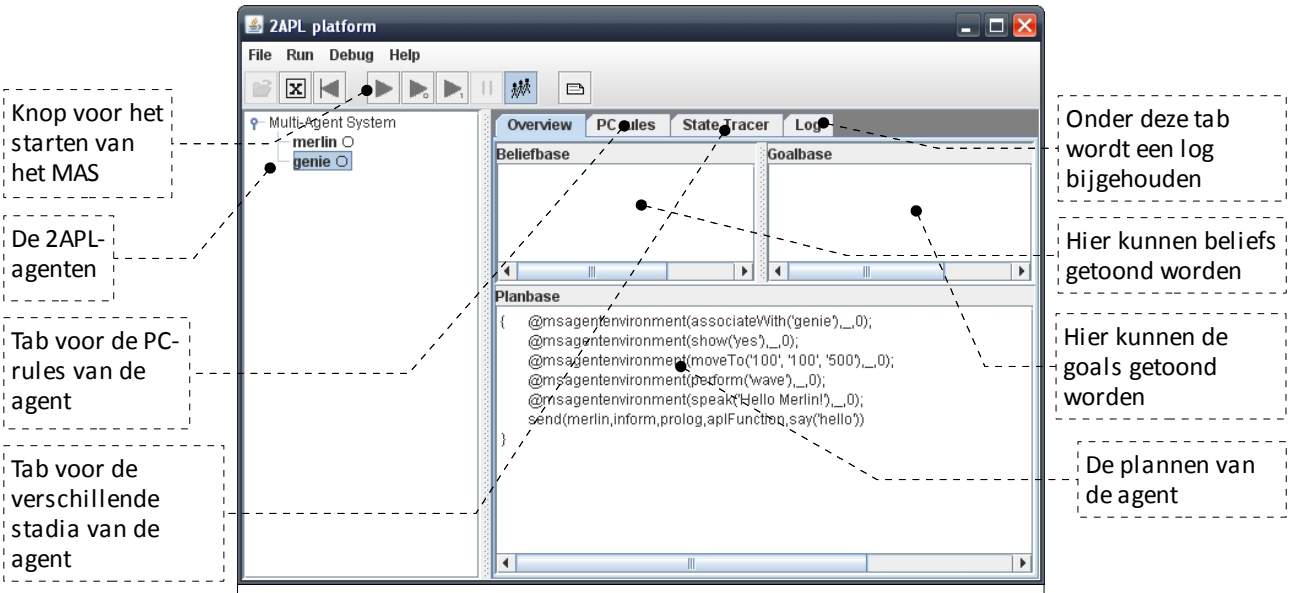
3 Michael Wooldridge is een professor aan de Universiteit van Liverpool en een bekend schrijver in de Agent Technology met maar liefst tweeduizend publicaties onder zijn naam betreffende dit onderwerp.[e]

2.2 2APL

In de inleiding hebben we al kennis gemaakt met de term 2APL. 2APL is de naam van de *agentgeörienteerde programmeertaal* die sinds enkele jaren in ontwikkeling is aan de Universiteit Utrecht. Bij een agentgeörienteerde programmeertaal denken we in agenten (in tegenstelling tot objecten zoals bij Java). Daarnaast is 2APL gericht op de ontwikkeling van Multi Agent Systems.
De naam 2APL, uit te spreken als "Double A P L", komt af van 2(AP)L; twee keer "AP" en een "L" en wordt soms ook geschreven als *APAPL.*. APAPL is een afkorting van "A Practical Agent Programming Language".

2APL is geschreven in Java en heeft dus ook de *JVM* nodig om te kunnen functioneren. Daarnaast maakt 2APL gebruik van het *Jade platform*[4] wanneer daar expliciet om gevraagd wordt. Jade maakt het mogelijk om een gedistribueerd MAS te ontwikkelen.

Hieronder staat een plaatje van het 2APL-frame:



Het 2APL-frame met een MAS geladen.

Wanneer een 2APL-programma geladen is, kan dit geëvalueerd worden met het 2APL-frame. Van elke afzonderlijke agent kan hiermee de huidige staat, maar ook voorgaande staten bekeken worden. Zodoende kan het geschreven MAS grondig getest worden, wat voor grotere systemen van fundamenteel belang is.
Het is echter ook mogelijk om een MAS te draaien *zonder* het hierboven afgebeelde 2APL-frame. Dit kan door gebruik te maken van de commandline:

2APL-MAS draaien zonder 2APL-frame	*Command line*
01	`java -jar 2apl.jar -nogui examples/virtual_recipe_assistant/virtualrecipeassistant.mas`

Het jar-bestand *2apl.jar* wordt op de gebruikelijke manier gestart en krijgt als parameters mee: *-nogui*, waarmee aangegeven wordt dat de GUI niet geladen dient te worden en vervolgens het volledige pad naar het te laden MAS. Door het 2APL-frame niet te laden zal het MAS aanzienlijk sneller draaien.

4 *Jade* staat voor *Java Agent Development Framework* en kan gebruikt worden om *FIPA-standaard-agents* te ontwikkelen. FIPA is een organisatie die standaarden voor agentgeörienteerde software ontwikkelt.

2.2.1 BDI-model

BDI is een afkorting van *Beliefs, Desires, Intentions*. Met dit model poogde prof. Michael Bratman te beschrijven hoe het menselijke rationele gedrag te verklaren is. Het BDI-model is fundamenteel afhankelijk van de volkpsychologie, waarbij gesteld wordt dat het mentale beeld dat de mens van de wereld heeft theorieën zijn.

Zoals hierboven gesteld is, bestaat het BDI-model uit drie componenten:

- **Beliefs;**
  De beliefs of *het geloof* van de agent beschrijft de staat van de agent zelf, de environment, of de staat van andere agenten. Er is gekozen voor de term "belief" en niet voor iets dergelijks als "knowledge", omdat de agent slechts kan *geloven* wat de staat van de environment en de andere agenten is. Een belief hoeft dus niet per definitie correct, of in overeenstemming met de werkelijkheid te zijn.
- **Desires;**
  De desires of *verlangens* van de agent geven de doelen van de agent aan. Er wordt wel gesproken van een "desirable state", oftewel: een te wensen toestand. Dat wil zeggen dat de agent wenst in een bepaalde toestand te verkeren of de environment in een bepaalde toestand te brengen. Een agent handelt vanuit zijn desires. Desires worden soms ook wel aangegeven met *goals* (*doelen*), bijvoorbeeld in 2APL. De term *goal* wekt meer de indruk dat de agent daadwerkelijke acties zal ondernemen dan de term *desire*.
- **Intentions.**
  De intentions of *intenties* van de agent beschrijven wat de agent zal doen om in zijn wenselijke toestand te komen; welke plannen gemaakt zullen worden.

2APL hanteert het BDI-model. Dit is zeer goed zichtbaar, omdat de beliefs, desires en intentions expliciet uitgeschreven (kunnen) worden in de broncode van 2APL-agenten. We zullen hier in in de volgende paragraaf dieper op ingaan.[f][g]

2.2.2 Syntax

De syntax van 2APL is voornamelijk gebaseerd op logica, maar heeft ook veel constructies van de populaire programmeertalen Java en C++ geërfd, zoals de "if-else" en de "while" constructies. 2APL gebruikt op veel plaatsen dezelfde syntax en regels als de programmeertaal *Prolog*[5].

Een 2APL-agent wordt gedefinieerd met *Beliefs, BeliefUpdates, Goals, Plans, PG-rules, PC-rules* en *PR-rules*. We zullen deze hieronder kort toelichten:

Beliefs

De beliefs van een 2APL-agent zijn gedefinieerd in *Prolog-facts* en *Prolog-rules*. Prolog facts zijn feiten[6] met een naam en één of meerdere parameters. Prolog-rules hebben twee Prolog-facts, waarvan de rechter geldt als pre-conditie; als de rechter Prolog-fact geloofd wordt, wordt de linker Prolog-fact ook geloofd. Beliefs hebben de vorm:

$pf = functor(p_0, \ldots, p_n).$ en $pr = prologfact$:- $prologfact.$, waarbij geldt dat $n > 0$. Wanneer we echter verwijzen naar een belief vanuit een andere plaats dan de beliefbase, gaat het altijd over de Prolog-fact vorm.

BeliefUpdates

Met behulp van BeliefUpdates is het mogelijk om de beliefs van de Agent bij te werken. Deze BeliefUpdates worden uitgeschreven met links de pre-condities, in het midden de naam van de update (te beginnen met een hoofdletter) en rechts de postconditie. Dus:

$BeliefUpdate = \{b_0, \ldots, b_n\} < BeliefUpdate\text{-}Name > (pf_0, \ldots, pf_{n2}) \{b_0, \ldots, b_{n3}\}$, waarbij b een belief voorstelt en $n \geq 0 \wedge n2 \geq 0 \wedge n3 > 0$.

5 Prolog is een programmeertaal, gebaseerd op de predicatenlogica. De naam staat dan ook voor "**Pro**gramming **log**ic". Prolog wordt veel gebruikt voor agent programmeertalen, omdat het duidelijk en gemakkelijk feiten kan weergeven. De lezers die geïnteresseerd zijn in Prolog kunnen voor meer info terecht op *www.learnprolognow.org*.

6 Laat hierbij gezegd zijn dat het woord "feit" eigenlijk niet de juiste term is, aangezien beliefs af kunnen wijken van de werkelijkheid.

Goals
De Goals van een agent zijn equivalent aan de desires uit het BDI-model. De notatie lijkt op die van de beliefs, met het verschil dat hier slechts Prolog-facts toegestaan zijn.

PG-rules
PG staat voor Planning Goal. Hierbinnen wordt gedefinieerd hoe de plannen gemaakt dienen te worden om een bepaalde goal te behalen. We kunnen een pg-rule als volgt definiëren:
$pgr = g_0, \ldots, g_n <- b_0, \ldots, b_{n2} | \{ \ldots \}$ waarbij g een goal is en op de stippellijn binnen de accolades de programmatuur komt te staan. De beliefs aan de rechterkant van de pijl vormen de pre-conditie van deze PG-regel.

PR-rules
PR-rules beschrijven de plannen die gemaakt worden wanneer een oorspronkelijk plan faalt, we spreken daarom dan ook van *Plan Repair rules*. Syntactisch gezien lijkt de PR-rule sterk op de PG-rule, met het verschil dat links geen goals gedefinieerd staan, maar plannen. Wanneer een dergelijke plan faalt en aan alle pre-condities voldaan wordt, zal de regel uitgevoerd worden.

PC-rules
De *Procedure Call rules* zijn in wezen functies zoals bekend in programmeertalen als C. Hier worden procedures geschreven die vanuit de PG-, PR- of de PC-rules aangeroepen kunnen worden. Recursie is mogelijk binnen de PC-rules. Een PC-rule ziet er als volgt uit:
$doThis(pf_0, \ldots, pf_n) <- b_0, \ldots, b_{n2} | \{ \ldots \}$ Net als bij de PG-rules zijn de beliefs aan de rechterkant van de pijl de pre-condities.

2.2.3 2APL-environment

In paragraaf 2.1.2 is beschreven wat de betekenis van een environment binnen de gehele Agent Technology inhoudt. De invulling die 2APL aan een environment gegeven heeft wijkt niet af van de eerder genoemde definitie.

Een 2APL-environment is een geheel van Java klassen waarvan 1 klasse aangewezen wordt als interface naar de 2APL-agenten, de zogeheten *main-klasse*. Deze klasse heeft methoden die een 2APL-agent aan kan roepen wanneer deze in de betreffende environment geplaatst is. Deze methoden kunnen, zoals het Java-methoden betaamt, return waarden geven. Daarnaast kan een 2APL-environment events sturen naar de agenten. De 2APL-interpreter verzorgt alle vertalingen van Java-typen naar 2APL-typen en vica versa.

In paragraaf 6.1.6 wordt uitvoerig besproken *hoe* een 2APL-environment gemaakt wordt volgens de *Environment Interface Standard*.

2.3 Microsoft Agent

De *Microsoft Agent* (of *MSAgent*) is een technologie ontworpen door Microsoft, welke het mogelijk maakt getekende karakters te besturen. Het is een opvolger van een oudere technologie: de *Microsoft Bob*. Met behulp van de zogenaamde *text-to-speech engine* kunnen de MSAgents tekst daadwerkelijk uitspreken in de taal die geïnstalleerd en geselecteerd is. Daarnaast kunnen de karakters elk een aantal animaties uitvoeren wat een grappige draai aan programma's kan geven.

Er zijn vier standaard karakters door Microsoft geïntroduceerd: Merlijn de tovenaar, Genie de geest uit de fles, Peedy de papegaai en Robby de robot. Daarnaast is het mogelijk om zelf Microsoft Agent karakters te maken, wat ervoor heeft gezorgd dat er een heleboel MSAgents over het gehele internet te vinden zijn.

Peedy

Een voorbeeld van een MSAgent karakter.

De meest bekende toepassing van de MSAgent technologie is te vinden in de oudere Microsoft Office pakketten. De hierin gebruikte agent is de overbekende bewegende paperclip die menigeen gebruiker geïrriteerd heeft (wat ook de reden is dat deze in de nieuwe Office pakketten niet meer aanwezig is). [h][i]

3 De MSAgentEnvironment

Een deel van de opdracht was om een aansprekende demo te maken. Een demo waar niet alleen de functionaliteiten van 2APL getoond worden, maar die ook een positieve indruk maakt op het publiek.
Mensen zijn visueel ingestelde wezens, daarom zou een geanimeerde karakter veel aansprekender zijn dan menig andere uitvoering van demo's. Zeker wanneer de animatie grappige kunstjes uit kan halen en bovendien hardop kan praten (je dus letterlijk aanspreekt). Mehdi kwam om deze reden met de opdracht om de Microsoft Agents te gebruiken voor de demo. Deze opdracht is uitgewerkt in het product de *MSAgentEnvironment*.

De MSAgentEnvironment is een simpele environment die gebruikt kan worden om de Microsoft Agents aan te sturen. Aan de MSAgentEnvironment kleven helaas twee nadelen:

☞ De MSAgentEnvironment draait enkel onder een Windows platform vanwege de Microsoft Agent technologie;

☞ Voorafgaand aan het gebruik van de MSAgentEnvironment dient het een en ander aan drivers geïnstalleerd te worden.

Desalniettemin zijn deze nadelen geen reden geweest om van het plan af te zien. De meeste gebruikers draaien een Windows systeem en wanneer goed gedocumenteerd wordt welke drivers geïnstalleerd moeten worden en waar deze te downloaden zijn, is ook het tweede nadeel gereduceerd tot acceptabel niveau.

3.1 Installatie van de MSAgentEnvironment

Zoals in voorgaande tekst vermeld staat is het noodzakelijk enkele drivers te downloaden en te installeren om de MSAgentEnvironment *in haar volledigheid* te benutten. In de vorige zin staat "in haar volledigheid" schuingedrukt, omdat het mogelijk is om delen van de functies van de MSAgentEnvironment achterwege te laten en niet te installeren, bijvoorbeeld de spraak. Bijlage C bevat het document dat de volledige installatie van de MSAgentEnvironment beschrijft.

Gebruikers die Windows ME, Windows 2000 of Windows XP draaien kunnen de installatie van enkele drivers overslaan, omdat deze standaard geïnstalleerd zijn bij deze operating systems. Het gaat daarbij om de Microsoft Core Components en het MSAgent karakter Merlin. De karakters Genie, Peedy en Robby dienen nog gedownload en geïnstalleerd te worden.
Om de MSAgents daadwerkelijk te laten praten met geluid, moet de *Speech API 4.0* geïnstalleerd worden. Dit is niet de nieuwste versie van de *SAPI*, maar de enige die werkt met de MSAgents. Daarnaast moet de *Learnout & Hauspie TruVoice Text to Speech Engine* geïnstalleerd worden. Deze driver bevat specifieke informatie voor een taal, bijvoorbeeld Amerikaans-Engels.
Wanneer al deze componenten geïnstalleerd zijn zal de MSAgentEnvironment volledig functioneren.

3.2 Environment Interface Standard

De MSAgentEnvironment is ontwikkeld conform de regels van de *Environment Interface Standard*, afgekort de *EIS.* Deze standaard definieert hoe een environment voor een agentgeörienteerde programmeertaal gebouwd moet worden en welke methoden in elk geval geïmplementeerd dienen te worden.
Er zijn meerdere agentgeörienteerde programmeertalen waarbij de environment geschreven dient te worden in pure Java, voorbeelden zijn 2APL, Jason en Jadex. Het doel van de Environment Interface Standard is om deze environments gelijk te trekken zodat een environment, oorspronkelijk ontwikkeld voor bijvoorbeeld 2APL, ook toegankelijk wordt voor bijvoorbeeld Jason-agenten.

3.2.1 EISification

Met de introductie van EIS is ook de term *EISification* geboren. EISification betekent dat een bestaande environment van een agentgeörienteerde programmeertaal dusdanig omgebouwd wordt dat deze voldoet aan de eisen die EIS stelt. In hoofdstuk 6.1.6 wordt dit nader toegelicht.

3.2.2 EIS documentatie

Bij een ge-EISificeerde environment hoort afdoende documentatie, omdat de agent-ontwikkelaar weet moet hebben van de mogelijk aan te roepen methoden en de mogelijk binnen te komen berichten. Om deze reden is er voor de EIS-standaard tevens een documentatie-standaard gedefinieerd.

Een EIS-document moet ten minste de volgende zaken bevatten:

- Een beschrijving van de environment;
- Het Jar-bestand welke de environment bevat en eventueel waar deze Jar gevonden kan worden;
- De entiteiten binnen de environment;
- De typen van de entiteiten;
- De mogelijk uit te voeren acties of methoden;
- De mogelijk te ontvangen berichten;
- De mogelijke acties om de environment zelf te besturen.

Bij het schrijven van de documentatie van een EIS-environment dient dus in feite alles vermeld te worden dat voor de agent-ontwikkelaar van belang is om de environment te kunnen gebruiken. [j]

3.3 Het gebruik

Hoewel in bijlage B de EIS-documentatie van de MSAgentEnvironment meegeleverd is, wordt het gebruik van de MSAgentEnvironment hieronder nogmaals beschreven ter volledigheid van dit document.

3.3.1 Initialisatie

Bij de initialisatie van deze environment kan worden aangegeven welke karakters geladen dienen te worden. Wanneer echter geen specifieke karakters gedefinieerd zijn, zullen deze alle vier automatisch geïnitialiseerd worden.

Waarom worden de karakters geladen bij de initialisatie van de environment en niet bij het starten? Het laden van een MSAgent karakter kan voor vertraging zorgen wat bij het starten van een environment minder mooi is. Daarnaast kunnen eventueel errors ontstaan bij het laden van de karakters. Het is wenselijk dit in een zo vroeg mogelijk stadium te ondervinden. Bovendien is het laden van een externe techniek principieel gezien het beste onder de initialisatie fase te plaatsen.

3.3.2 MSAgent tonen en laten verdwijnen

Wanneer een karakter van de MSAgent geïnitialiseerd is, is deze nog niet zichtbaar. De methode die aangeroepen moet worden om de MSAgent te laten verschijnen is $show(a)$, waarbij a aangeeft of de MSAgent geanimeerd moet verschijnen of "normaal". a kan hiertoe de tekst "*yes*" of "*no*" bevatten.
Evenzo werkt de methode om de MSAgent te laten verdwijnen. Deze methode heeft de vorm $hide(a)$.

3.3.3 Tekstballon laten verschijnen en praten

De MSAgentEnvironment kan een bepaald karakter laten praten. De methode die hiervoor aangeroepen dient te worden is $speak(s)$ waarbij s een string is; de tekst die uitgesproken dient te worden. Bij het uitvoeren hiervan zal een tekstballon verschijnen met de aangegeven tekst erin. Ook zal de Agent deze tekst hardop uitspreken, mits de benodigde drivers hiervoor geïnstalleerd zijn natuurlijk.
Wanneer de methode $think(s)$ wordt aangeroepen zal een ander soort tekstballonnetje verschijnen in de vorm van een wolkje. In cartoons wordt hiermee aangegeven wat een karakter denkt en dat is hier ook het geval. De tekst s die de MSAgent "denkt" wordt hierbij niet uitgesproken.

3.3.4 Uitvoeren van animaties

Elk karakter heeft een vast aantal animaties die deze uit kan voeren. In de Javadoc van de MSAgentEnvironment staan deze genoteerd als publieke attributen in de klassen van de karakters. Om een MSAgent een animatie uit te laten voeren dient de volgende methode te worden gebruikt: $perform(s)$, waarbij s de naam van de animatie is in de vorm van een string.

Animatie

De animatie "domagic1" van Peedy

3.3.5 Verplaatsen

Een MSAgent kan door de gebruiker met de muis naar elke willekeurige locatie op het scherm verplaatst worden. Echter kan de MSAgent ook zelf verplaatsen met behulp van de methode $moveTo(x, y, d)$. De parameters x en y zijn hierbij de coördinaten van de gewenste locatie op het scherm en d is de vertraging, dus hoe lang de MSAgent er over moet doen om zich naar de gegeven locatie te verplaatsen. Wanneer de MSAgent zichtbaar is gemaakt zal deze geanimeerd verplaatsen, bijvoorbeeld door met de vleugels te wapperen (Peedy). Wanneer daarentegen de MSAgent niet zichtbaar is kan deze ook verplaatst worden, de parameter d heeft in dat geval geen invloed.

3.3.6 De MSAgent laten wijzen

Hoewel in Nederland wijzen niet altijd als netjes beschouwd wordt, heeft de Microsoft Agent de mogelijkheid om te wijzen. Hiervoor dient de methode $gestureAt(x, y)$ gebruikt te worden, waarbij hier, net als bij het verplaatsen, de parameters x en y de coördinaten op het scherm zijn. Helaas beschikt de MSAgent niet over de capaciteit om heel gericht te wijzen, maar zal deze slechts naar boven, beneden, links of rechts wijzen, afhankelijk van zijn eigen positie en de opgegeven coördinaten.

4 Virtual Recipe

Om een computerprogramma te laten assisteren bij het koken van een recept, zal het programma vanzelfsprekend kennis moeten hebben van het betreffende recept. Recepten zoals wij die kennen zijn normaal gesproken geschreven in zinnen als: "voeg nu twee snufjes zout toe en roer de boel een beetje door". Dergelijke taal is voor een mens heel logisch en dus zonder meer begrijpelijk. Een computerprogramma of een software agent daarentegen zal eerst een stevige fundering ingeprogrammeerd moeten hebben om de woorden te kunnen vertalen naar de bekende nullen en enen; begrijpend lezen is een ware kunst! De studie die hierover gaat heet *Taal en Spraak Technologie*.
Het vergt weinig technische kennis om in te kunnen zien dat het herkennen van recepten in zinnen als hierboven onmogelijk binnen de scope van een halfjarig project kan vallen. Er is dus een alternatieve notatie van recepten nodig, of anders gezegd: er is een *exacte notatie* van recepten nodig.

4.1 Virtual Recipe in detail

Voor de Virtual Recipe Assistant is een bescheiden notatie van recepten bedacht: de zogenaamde *Virtual Recipe*. Dit is een manier waarop een recept in exacte termen beschreven kan worden, wat een recept bruikbaar maakt voor hedendaagse computerprogramma's. Deze schrijfwijze voor recepten omvat de meest gangbare activiteiten die uitgevoerd moeten worden bij het bereiden van een maaltijd. Hierbij moet worden gedacht aan:

- Het selecteren van pannen en schalen en dergelijke;
- Het toevoegen van ingrediënten in de pannen en schalen met een bepaalde hoeveelheid;
- Het uitvoeren van simpele acties zoals roeren.

Er is voor gekozen om binnen de Virtual Recipe de factor "tijd" buiten beschouwing te laten, omdat de virtuele recepten anders te ingewikkeld worden om uit te lezen met The Virtual Recipe Assistant en te "begrijpen" door de agent.

4.1.1 Actions

Een Virtual Recipe bestaat uit een reeks handelingen die uitgevoerd moeten worden, zo'n handeling noemen we een *Action.* Deze Actions hebben betrekking op ingrediënten: *Ingrediënts* en/of op het gereedschap: *Tools*.
Voor de Virtual Recipe Assistant worden drie soorten Actions gebruikt:

- **Select-action;**
  Hierbij dient een Tool geselecteerd te worden. Het gaat in dit geval om een *Container-tool*[7]. Een dergelijke Action heeft dus de vorm: $select(<Container\text{-}tool>)$.
- **Insert-action;**
  Hierbij dient een bepaalde Ingrediënt in een bepaalde hoeveelheid toegevoegd te worden aan een Tool. Een Insert-action heeft de volgende vorm:
  $insert(<Ingrediënt>,<hoeveelheid>,<Container\text{-}tool>)$.
- **Perform-action.**
  Hierbij wordt een bepaalde handeling gedaan met behulp van een Tool. Hiervan is de vorm:
  $<Action>(<Tool>)$.

7 Zie paragraaf 4.1.2

4.1.2 Tools

In de paragraaf hierboven hebben we al kennis gemaakt met de term Tool. We hebben in een voorbeeld kunnen zien dat een Tool bijvoorbeeld een pan zou kunnen zijn. Een Tool kan ook een lepel zijn, een spatel, een mes of een kaasschaaf. Toch zit er een essentieel verschil tussen deze drie voorbeelden van Tools. Een pan wordt namelijk gebruikt om ingrediënten in te doen, terwijl een mes gebruikt wordt om een handeling uit te voeren op een ingrediënt. Om deze reden maken we onderscheid tussen *Container-tools* en *Executive-tools.*

Je kunt van een Executive-Tool specificeren welke acties er mee uitgevoerd kunnen worden, zo kun je bijvoorbeeld bepalen dat je kunt roeren met een vork en met een lepel. Als dan de agent de instructie geeft om te roeren zonder aan te geven met welke Tool, dan kan elke Tool gebruikt worden die deze actie ondersteunt.

Wanneer we een Container-tool willen definiëren, zullen we moeten vaststellen hoe groot deze Tool moet zijn; de capaciteit (*Capacity*) van de Container-tool. De Capacity geeft met een getal aan hoeveel Ingrediënts in de Tool gedaan kunnen worden.

Van een Tool is het uiteindelijk mogelijk om de volgende parameters te definiëren:

- **name;**
  De *name* parameter wordt gebruikt om te refereren naar een specifieke Tool.
- **capacity;**
  Alleen in het geval van een Container-tool.
- **title;**
  Dit is de representatieve naam van de tool, hier mogen bijvoorbeeld ook spaties in zitten.
- **image;**
  Eventueel kan aangegeven worden hoe de Tool er uitziet.
- **actions.**
  Alleen in het geval van een Executive-tool dient aangegeven te worden welke handelingen met de Tool gedaan kunnen worden.

Wanneer van een Tool de parameter "capacity" gedefinieerd is, is er automatisch sprake van een Container-tool. Deze Tool heeft dan ook automatisch de speciale Action *insert*, dit hoeft dus niet meer apart aangegeven te worden. De Action *insert* houdt in dat een Ingrediënt in de Container-tool gedaan kan worden.

4.1.3 Ingrediënts

Ingrediënts kunnen in een bepaalde hoeveelheid toegevoegd worden aan de inhoud van een Container-tool.
Een Ingrediënt heeft, net als een Container-tool, een bepaalde Capacity. In dit geval gaat het om de minimaal benodigde capaciteit van een Container-tool om dit Ingrediënt te kunnen bevatten:

We definiëren T als alle Container-tools, I als alle Ingrediënts en $C(x)$ als de capaciteit van x . We kunnen dan stellen dat: $\forall i \in I\,(C(i) > C(t \in T) \rightarrow i \notin t)$.
Zoals uit deze formule af te lezen is, kan een ingrediënt niet toegevoegd worden aan een Container-tool wanneer de benodigde capaciteit van dit ingrediënt groter is dan de capaciteit van de tool.

Verder zijn er voor een Ingrediënt de volgende parameters mogelijk:

- **name;**
  De *name* parameter wordt gebruikt om te refereren naar een Ingrediënt.
- **capacity;**
  De c*apacity* geeft de benodigde capaciteit aan, zoals hierboven uitgelegd is.
- **title;**
  Dit is de representatieve naam van de Ingrediënt.
- **image;**
  Eventueel kan aangegeven worden hoe de Ingrediënt er uitziet.
- **unit;**
  De *unit* geeft aan welke *meeteenheid* gebruikt moet worden bij deze Ingrediënt, voorbeelden zijn "gram" voor macaroni en "snufjes" voor zout.
- **amounts.**
  Met de parameter a*mounts* geef je aan in welke hoeveelheden we moeten denken voor een bepaald Ingrediënt. Het is bijvoorbeeld handiger om aan te geven dat water per 50 ml. toegevoegd kan worden dan per 1 ml. of per 1 l.

4.1.4 Definitie van een Virtual Recipe

Met het onderscheiden van de verschillende onderdelen van een recept kunnen we op een algemene definitie van een recept komen. Een Virtual Recipe bestaat uit een reeks Actions die betrekking hebben op Tools en Ingrediënts.
De formele definitie van een Virtual Recipe v is dus simpelweg $v = [a_1, \ldots, a_n]$ waarbij $n > 0$.
Deze acties van het recept worden achtereenvolgend uitgevoerd.

Stel we hebben het volgende recept voor het bereiden van 100 ml. kokend water:

"Plaats een steelpan op het fornuis. Giet hier 100 ml. water in en zet vervolgens het vuur aan. Wacht enkele minuten tot het water begint te borrelen en zet vervolgens het vuur weer uit."

en we willen dit recept omzetten in een Virtual Recipe. Wat we dan doen is het onderscheiden van de handelingen die een Virtual Recipe ondersteunt, dat zijn handelingen die in te delen zijn in de drie mogelijke Actions van een Virtual Recipe[8]. Voor dit voorbeeld zijn dat:

- $a_1 = select\,(steelpan)$;
- $a_2 = insert\,(water, 100, steelpan)$;
- $a_3 = turn\,on\,(fire)$;
- $a_4 = turn\,off\,(fire)$.

Onze Virtual Recipe is dan simpelweg $R[a_1, a_2, a_3, a_4]$.

8 Dat zijn de *Select-action*, *Insert-action* en de *Perform-action*, zoals gedefinieerd in paragraaf 4.1.1 .

4.2 Gebruik van XML

Wanneer de Virtual Recipe Assistant draait is een Virtual Recipe uiteindelijk in drie vormen aanwezig: *Java*, *Prolog* en *XML*. Deze laatste vorm is de initiële vorm van een Virtual Recipe; een virtueel recept wordt in XML gedefinieerd van waaruit vervolgens Java objecten en Prolog code gegenereerd wordt.

4.2.1 Waarom XML?

De keuze voor XML is te onderbouwen aan de hand van *zes* factoren die meespelen. Deze factoren staan hieronder benoemd.

Er is gekozen voor XML als uitgangspunt, omdat

a) hiermee *dynamisch* nieuwe onderdelen toegevoegd kunnen worden (in tegenstelling tot statische Java objecten);
b) hiermee de data zoveel mogelijk van de code gescheiden kan blijven;
c) XML een ideale constructie heeft om de eigenschappen per onderdeel te kunnen beschrijven;
d) het gebruik van XML de uitbreidbaarheid van het programma ten goede doet komen;
e) XML een zeer bekende taal is, waardoor de meeste ontwikkelaars weinig moeite zullen hebben met het schrijven van een Virtual Recipe;
f) XML overzichtelijk en gemakkelijk uit te lezen is.

We vervolgen dit hoofdstuk door per onderdeel te beschrijven hoe dit gespecificeerd dient te worden.

4.2.2 Tools definiëren

De Tools die in een recept gebruikt worden moeten ergens gedefinieerd worden. Er moeten immers parameters gespecificeerd worden, zoals we in paragraaf 4.1.2 hebben kunnen zien.

Er is voor gekozen om de Tools en Ingrediënts apart van de recepten zelf te definiëren. Zodoende kunnen Tools en Ingrediënts hergebruikt worden en is er geen sprake van redundantie. Daarnaast zouden de recepten vrijwel onleesbaar worden vanwege alle parameters die bij de Tools en Ingrediënts horen.

Het definiëren van Tools gebeurt in het bestand *tools.xml* dat in de map *xml* geplaatst is. Een Tool dient binnen de *tools* root-tag geplaatst te worden. De naam van de tag geldt als de parameter *name* van de Tool.

In de tabel hieronder staan de parameters voor een tool-tag:

Parameters - tool-tag			
Naam	**Verplicht**	**Formaat waarde**	**Korte beschrijving**
title	nee	string	representatieve naam voor de Tool.
image	nee	string	pad naar het bestand.
capacity	nee	integer	hoeveelheid aan tool toe te voegen eenheden.

Wanneer er sprake is van een Executive-tool, kan worden gespecificeerd welke acties met de tool gedaan kunnen worden. Dit is direct de manier waarop Actions gedefinieerd worden. Actions worden dus niet in een apart XML-bestand gedefinieerd, omdat dit redundant zou zijn aangezien er voor Actions verder geen inhoudelijke parameters te definiëren zijn. De enige parameters die een Action kan hebben zijn de *name* ter verwijzing en de *title* als representatieve naam.

Voorbeeld van Tools in XML *XML code*

```
<tools>
  <big_pan title='big pan' image='big pan.jpg' capacity='15000' />
  <spoon image='spoon.jpg'>
    <stir />
    <taste />
  </spoon>
</tools>
```

Hierboven is een voorbeeld te zien van hoe Tools gedefinieerd kunnen worden. We zien een Container-tool betiteld als "big pan" waar kennelijk 15 000 ml in past. Daaronder vinden we een Executive-tool, genaamd "spoon" waarmee we kennelijk kunnen roeren en proeven.

4.2.3 Ingrediënts definiëren

De Ingrediënts staan gedefinieerd in het bestand *ingredients.xml* in dezelfde map als waar ook de tools staan. De opbouw van het XML-bestand lijkt op die van de tools: binnen de root-tag *ingredients* staan de ingrediënten opgesomd waarvoor geldt dat ook hier de tag-naam geldt als de naam van het ingrediënt.

De tabel hieronder weergeeft de te gebruiken parameters voor een ingredient-tag:

Parameters - ingredient-tag

Naam	**Verplicht**	**Formaat waarde**	**Korte beschrijving**
unit	ja	string	meeteenheid van het ingrediënt.
amounts	ja	integer	stapgrootte van de hoeveelheid van het ingrediënt.
capacity	ja	double	benodigde capaciteit voor de gedefinieerde hoeveelheid.
image	nee	string	pad naar het bestand.

Met de parameter *amounts* is aan te geven in welke hoeveelheden we moeten denken voor het betreffende ingrediënt; de stapgrootte van de toename van de hoeveelheid. In paragraaf 5.1.3 van het volgende hoofdstuk is te lezen dat de gebruiker de hoeveelheid van een ingrediënt kan manipuleren, hierbij wordt deze stapgrootte gebruikt.

Voorbeeld van Ingredients in XML *XML code*

```
<ingredients>
  <water unit='ml' amounts='50' capacity='50' image='water.jpg'/>
  <macaroni unit='gram' amounts='50' capacity='40'/>
</ingredients>
```

In het voorbeeld hierboven staan twee ingrediënten gedefinieerd. Hieruit is bijvoorbeeld af te lezen dat er een capaciteit van 40 ml. nodig is om 50 gram macaroni te bevatten. Hierbij is overigens geen rekening gehouden met de werkelijke massa van de betreffende ingrediënten.

De Tools en Ingrediënts die gedefinieerd zijn, zullen verschijnen in de "kast" Virtual Recipe Assistent[9]. Wanneer voor een Tool/Ingrediënt een image gedefinieerd is, wordt deze hier getoond, anders verschijnt er een wit vierkant. Beschouw de gedefinieerde Tools en Ingrediënts als door-de-user-mogelijk-te-gebruiken entiteiten voor een door-de-agent-gegeven instructie.

9 Dit is zichtbaar in paragraaf 5.1.2 .

4.2.4 De recepten

In de directory *xml/recipies* staan alle recepten gedefinieerd. Zodra hier een XML-bestand in geplaatst wordt, zal deze uitgelezen worden bij het starten van de Virtual Recipe Assistant.
Een recept moet een root-tag hebben genaamd "recipe". Deze root-tag moet bovendien een parameter "name" bevatten, zodat naar dit recept gerefereerd kan worden. De parameter "title" komt ook hier weer om de hoek zetten en is ook hier niet verplicht om te gebruiken.
Een recept moet een tag genaamd "actions" bevatten waarbinnen alle Actions gerangschikt staan in de volgorde waarop ze uitgevoerd dienen te worden. Elke Action die hier genoteerd staat moet gedefinieerd zijn in de tools.xml, anders zou de assistent een niet uit te voeren instructie moeten geven wat natuurlijk niet de bedoeling is.

In paragraaf 4.1.4 stond een voorbeeld van de definitie van een Virtual Recipe. Deze is hieronder in XML uitgewerkt:

Voorbeeld van Virtual Recipe in XML *XML code*

```
01  <recipe name='hot_water' title='hot water'>
02    <insert>
03      <steelpan />
04      <water amount='100' />
05    </insert>
06    <actions>
07      <turn_on comment='People like their food hot.'>
08        <fire />
09      </turn_on>
10      <turn_off comment='We do not want any trouble with firemen.'>
11        <fire />
12      </turn_off>
13    </actions>
14  </recipe>
```

Oplettende lezers hebben wellicht opgemerkt dat de Action $select(steelpan)$ niet in bovenstaand XML-bestand gedefinieerd staat. Dit zou overbodige informatie geweest zijn, omdat al te zien is dat we een steelpan nodig hebben wanneer we de *insert*-Action gebruiken. De assistent houdt hier rekening mee en zal, voordat hij de instructie geeft om een ingrediënt toe te voegen, de opdracht geven de Container-tool te selecteren.

Op de regels 7 en 10 is commentaar aan de actie toegevoegd. Commentaar kan bijvoorbeeld gebruikt worden om een uit te voeren handeling nader toe te lichten of simpelweg om het dialoog wat vloeiender te laten verlopen.

Bekijk nu onderstaand voorbeeld:

Voorbeeld gebruik van zelf te bepalen tools *XML code*

```
01  <recipe name='example'>
02    <requirements>
03      <tool id='0' capacity='1100' />
04    </requirements>
05    <actions>
06      <insert>
07        <tool id='0' />
08        <water amount='100' />
09      </insert>
10      <stir>
11        <tool />
12      </stir>
13    </actions>
14  </recipe>
```

Hier zien we op de regels 2 t/m 3 dat er een zogenaamde *requirement* gedefinieerd wordt. Door hier een requirement te definiëren met een bepaalde *id* en *capacity* is het mogelijk deze verderop in het recept te benutten, dit is te zien op regel 7 van het voorbeeld. Door deze constructie te gebruiken kun je aangeven dat er een Container-tool nodig is met ten minste de gespecificeerde capaciteit. De kok mag in dat geval dus zelf een pan uitkiezen, zolang deze aan de gestelde voorwaarde voldoet.
Op regel 10 zien we een tag genaamd "tool" zonder id. Dit geeft aan dat elke Tool gebruikt mag worden. Natuurlijk heeft de gebruiker slechts de keuze uit de Tools die de betreffende Action (in dit geval, "stir") ondersteunen.

4.2.5 Recepten encapsuleren

Het is mogelijk om binnen een Virtual Recipe andere Virtual Recipes te encapsuleren. Met deze manier van *generalisatie* hoeven vaak-terugkerende handelingen, bijvoorbeeld het koken van water en dergelijke, niet meer dan één maal gespecificeerd te worden.
Let wel dat bij het includen van een recept, *slechts de acties* geëncapsuleerd worden. De zogenaamde requirements, waar we in de vorige paragraaf kennis mee gemaakt hebben, worden *niet* meegenomen. De reden hiervoor is dat er botsingen zouden kunnen komen wanneer in meerdere recepten dezelfde ID's gebruikt zijn.

Voorbeeld van een include *XML code*

```
01  <recipe name='macaroni_bolognese' title='macaroni bolognese'>
02
03    <requirements>
04      <tool id='0' capacity='2000' />
05    </requirements>
06
07    <actions>
08
09      <include recipe='plain_macaroni' />
10
11  ...
```

In bovenstaand voorbeeld wordt op lijn 9 een ander recept geinclude. We zien dat de tag *include* een parameter *recipe* heeft met de waarde van de naam van een recept. Het is mogelijk dat een geinclude recept zelf ook includes doet.

5 De Virtual Recipe Assistant

De kookkunst is een zeer oude en cultuurafhankelijke, culinaire aangelegenheid. Door de eeuwen heen heeft de kookkunst verschillende ontwikkelingen doorgemaakt. Zo begonnen de oude Grieken met de kunstzinnige vorm van het koken, waarna in de middeleeuwen de nadruk veel meer gelegd werd op het eten van vlees en de kwantiteit hiervan. In de Renaissance bloeide de aandacht voor de kookkunst pas weer op.
Vandaag de dag verschilt het sterk per bevolkingsgroep in hoeverre er aandacht aan het koken besteed wordt. Studenten bijvoorbeeld nemen veelal eerder genoegen met een simpele, snelle (en vooral goedkope) maaltijd dan een doorsnee huishouden. Vaak zijn dit dan ook de beginners in de kookkunst. Voor deze beginners met een matige kennis van het bereiden van verschillende recepten is de *Virtual Recipe Assistant* een handige uitkomst.[k]

De Virtual Recipe Assistant is een applicatie waarbij de gebruiker een virtueel recept kookt en daarbij begeleid wordt door een assistent. De assistent zal er op toezien dat de gebruiker de juiste handelingen verricht voor een virtueel recept naar keuze.

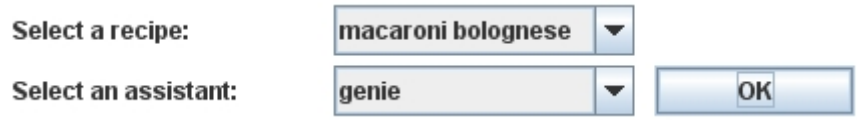
De Virtual Recipe Assistant is een combinatie van een ge-EISificeerde environment met een 2APL-agent. Er is dus geen sprake van een Multi Agent System, gezien er slechts één agent betrokken is bij deze applicatie. Deze applicatie maakt gebruik van de eerder beschreven MSAgentEnvironment. De assistent heeft de vorm van een Microsoft Agent karakter naar keuze.

5.1 De werking

De werking van de Virtual Recipe Assistant zal hieronder per onderdeel beschreven worden.

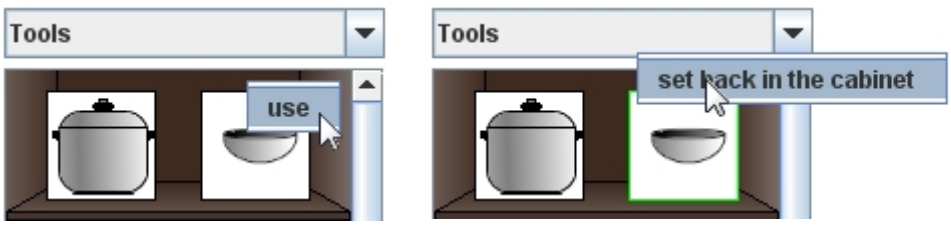
5.1.1 Assistent en recept kiezen

Zodra de Virtual Recipe Assistant gestart is, dient een keuze gemaakt te worden uit de vier MSAgent karakters. Daarnaast moet het gewenste recept geselecteerd zijn, waarna de gebruiker op de knop moet drukken om verder te gaan. Na deze handelingen zal de agent verschijnen en beginnen met het geven van instructies. Er kunnen voordien geen handelingen met betrekking op het recept gedaan worden; alle overige functionaliteiten zijn ge-disabled.

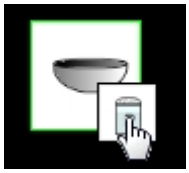


5.1.2 Container-tool (de)selecteren

Het eerste dat de virtuele assistent de gebruiker zal opdragen is het selecteren van een Container-tool met een bepaalde capaciteit. Dit selecteren gebeurt door op de gewenste Tool te klikken waarop vervolgens een klein menu'tje verschijnt. Kies de optie "use", zodat deze tool in het grote zwarte vlak verschijnt. Wanneer een Container-tool geselecteerd is heeft deze een groene rand om de afbeelding wat aangeeft dat deze in gebruik is. Door op de Container-tool te klikken kan deze ge-deselecteerd worden.

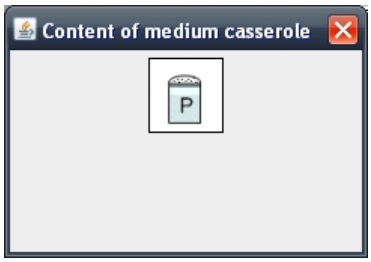


5.1.3 Ingrediënten toevoegen en verwijderen

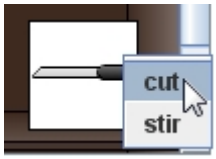


Een ingrediënt kan geselecteerd worden door op het icoon in de voorraadkast rechts te klikken. Er verschijnt dan een kleiner icoontje dat met de muis meebeweegt. Met behulp van de scroller van de muis kan de kwantiteit van het geselecteerde ingrediënt gemanipuleerd worden. Wanneer vervolgens de muis met het ingrediënt over de Container-tool in het zwarte deel van de applicatie beweegt, zal deze Container-tool een groene lijn om zich heen krijgen. Door vervolgens te klikken wordt het ingrediënt toegevoegd, maar alleen als deze nog past. Past deze niet meer in de Container-tool, dan gebeurt er niets en blijft het ingrediënt geselecteerd totdat de gebruiker elders klikt om het ingrediënt los te laten.

Wanneer het ingrediënt is toegevoegd, wordt de inhoud van de Container-tool zichtbaar in een popup-venstertje. Dit venstertje kan weggeklikt worden door op het rode kruisje te klikken en kan op enig willekeurig moment weer tevoorschijn gehaald worden, door op de Container-tool te klikken.
Als de gebruiker op een ingrediënt in dit venster klikt, verschijnt er een popup-menu met de optie "Remove", zodat het ingrediënt weer verwijderd kan worden. Hoewel deze mogelijkheid wellicht minder realistisch is, is deze toegevoegd om foutjes te kunnen corrigeren.



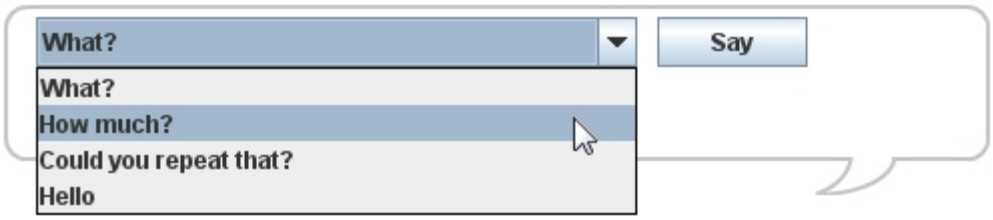
5.1.4 Uitvoeren van acties



Wanneer de agent een instructie geeft om een bepaalde actie uit te voeren zoals "roer het geheel" of "snij de tomaat", dient hiervoor een gereedschap gebruikt te worden, bijvoorbeeld een mes. Bij de Virtual Recipe Assistant gebeurt het uitvoeren van acties door op een gereedschap te klikken waarna een popup-menu verschijnt zoals ook bij voorgaande paragrafen. De gebruiker heeft dan keuze uit de acties die in dit menu staan. Door simpelweg een actie aan te klikken wordt deze uitgevoerd.

5.1.5 Communiceren met de agent

Tijdens het bereiden van een virtuele maaltijd is het mogelijk om te communiceren met de agent. Hiervoor kan de gebruiker uit enkele voorgedefinieerde zinnen kiezen in de selectbox onderaan het programma. De keuze aan zinnen die de gebruiker tot zijn/haar beschikking heeft is situatie-afhankelijk. Zo is het bijvoorbeeld mogelijk te vragen hoeveel van een bepaald ingrediënt ook al weer aan de Container-tool toegevoegd moest worden nadat de agent hiertoe de instructie had gegeven.



5.2 De assistent

De assistent die de gebruiker begeleidt bij het koken is een 2APL-agent. Zodra de Virtual Recipe Assistant gestart is, zal deze agent alle informatie van de beschikbare Tools en Ingrediënts opvragen bij de enviroment. De specifieke informatie van een recept wordt pas opgevraagd zodra een recept geselecteerd wordt.
De assistent heeft een lijst met instructies in zijn beliefbase en houdt bij welke instructies gegeven zijn, evenals welke acties al door de gebruiker uitgevoerd zijn.

5.2.1 Willekeur

Om de assistent ietwat afwisselender te maken is op enkele plaatsen sprake van willekeur; de meeste tekst die de agent spreekt wordt op willekeurige basis bepaald. Er zijn bijvoorbeeld zeven verschillende zinnetjes die de assistent uit kan spreken wanneer de gebruiker een instructie correct opgevolgd heeft.

5.2.2 Model Agent-Based Speech Act Generation

In de doelstelling is gesteld dat in het op te leveren eindproduct het model *Agent-Based Speech Act Generation* verwerkt moet zijn. Dit model schetst hoe menselijke conversaties door agenten gedaan kunnen worden, waarbij de focus gelegd is op het afhandelen van fouten binnen de conversatie.

Fouten afhandelen
Wanneer we een dialoog indelen met een zender en een ontvanger, is er sprake van een doel bij de zender: het overbrengen van een bepaalde boodschap. Om dit doel te bereiken gaat de zender met de ontvanger een dialoog aan, waarbij de boodschap omgezet is in één of meerdere zinnen.
Echter kunnen bij het voeren van een dergelijk dialoog veel fouten ontstaan. Onder een fout wordt hier verstaan: elke afwijking in nauwkeurigheid of correctheid waarbij het verloop van het gesprek afwijkt van de planning. Bij het aantreffen van een fout dienen stappen ondernomen te worden om het dialoog zo snel mogelijk terug te brengen in de juiste staat.

Binnen de scope van het model worden de volgende fouten onderscheiden:

- **Inconsistenties tussen de beliefs van de zender en de ontvanger;**
  De zender en ontvanger hebben een verschillend beeld van een bepaald feit. In het proefschrift van Nieske wordt het volgende voorbeeld gegeven:
  - [*Z]: Wat wil je eten?*
  - *[O]: Iets met vlees.*
  - *[Z]: Kip Siam?*
  - *[O]: Ik wil iets met vlees.*
  - *[Z]: Dat is met vlees.*
  - *[O]: Nee dat is met kip.*

  Kennelijk heeft O in dit voorbeeld een ander beeld van kip dan Z, er is sprake van inconsistentie. Bij het ondervinden van inconsistenties tussen de beliefs dienen deze zo snel mogelijk opgelost te worden. Hierbij zal een partij zijn beliefs aan moeten passen.
- **Afwijkingen van het dialoog-model.**
  De ontvanger handelt op een onverwachte manier en wijkt dus af van het model. Voorbeeld:
  - *[Z]: Wat wil je eten vanavond?*
  - *[O]: Pasta pesto!*
  - *[Z]: Wil je rode pesto?*
  - *[O]: Ik wil pasta!*
  - *[Z]: Er is rode en er is groene pesto...*
  - *[O]: Eh, ik wil groene pesto.*

  De ontvanger beantwoordt de vraag van de zender niet, maar herhaalt wat hij eerder zei. De zender legt vervolgens uit waarom deze informatie nodig is om alsnog de benodigde informatie te verkrijgen.

Het model in beeld

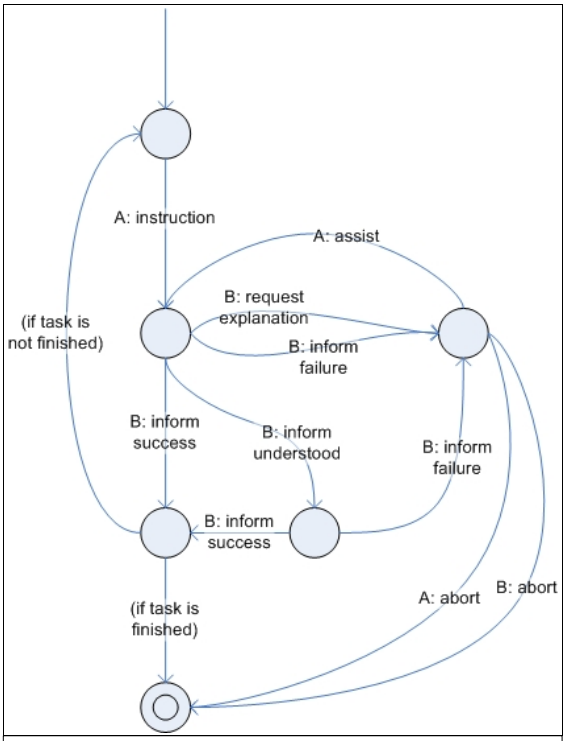
De figuur hiernaast is ontleend aan het proefschrift van Nieske en visualiseert hoe een taak-geörienteerd dialoog verloopt waarbij deelnemer A instructies geeft aan deelnemer B.

Het dialoog begint met A die een instructie geeft aan B, te zien aan de pijl tussen de bovenste cirkel en deze eronder. B kan hierop verschillend reageren:

1. Vragen om extra uitleg;
2. Aangeven dat de actie mislukt is;
3. Aangeven dat de actie gelukt is;
4. Aangeven dat de instructie begrepen is.

In de eerste twee gevallen zal A moeten assisteren. We hebben dit in voorgaande voorbeeld-dialogen kunnen zien, bijvoorbeeld: "Er is rode en er is groene pesto", waarmee een eerder gestelde vraag van motivatie wordt voorzien.
In het derde geval is de instructie gelukt en kan A doorgaan met een volgende instructie, als het dialoog nog niet afgelopen is.
In het vierde geval zegt B de instructie begrepen te hebben, maar kan dit alsnog in mislukking uitlopen.

Zowel deelnemer A als B kan bij elke mislukking besluiten het dialoog te verlaten.



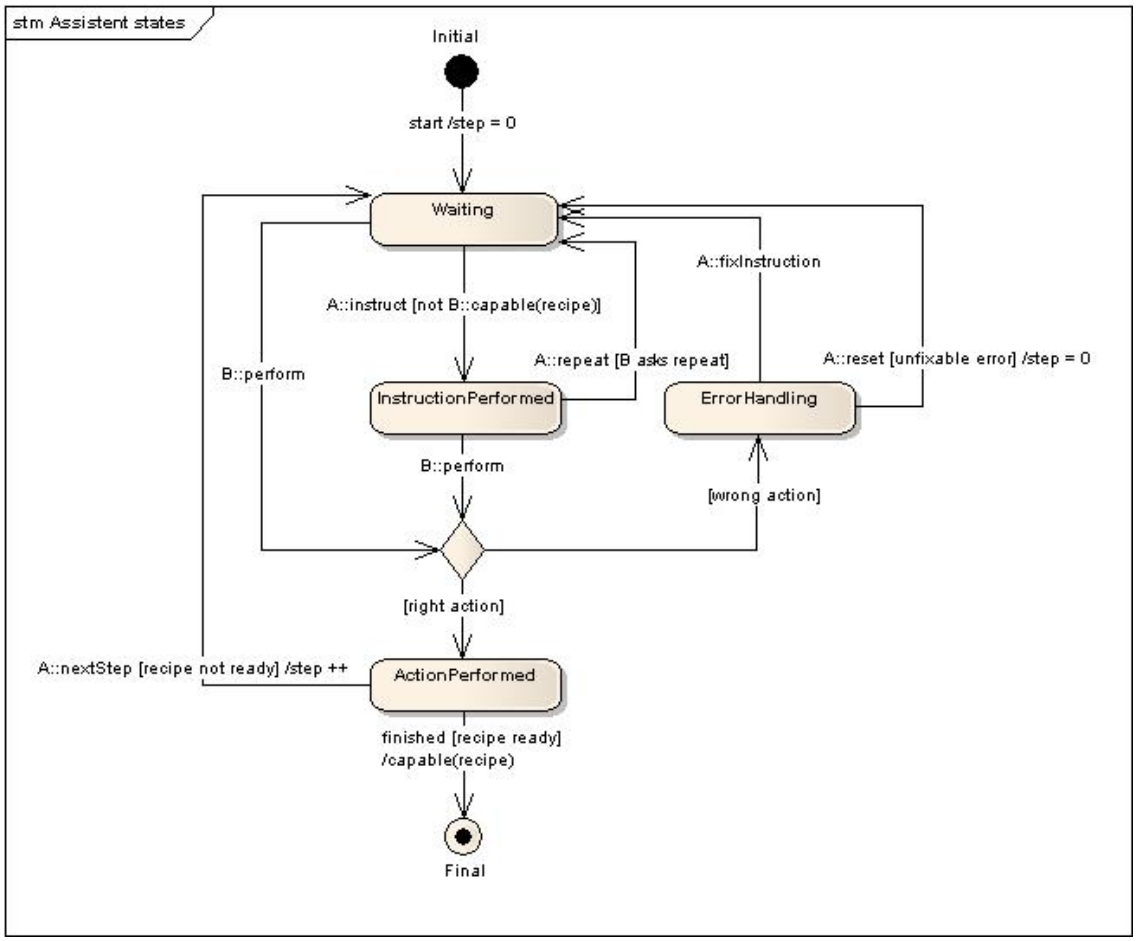
Een generiek taak-geörienteerd dialoog, waarbij deelnemer A deelnemer B instrueert tot het uitvoeren van een zekere taak.

Capabilities

Wanneer A een instructie geeft aan B, is A kennelijk overtuigd van de haalbaarheid van de uit te voeren actie door B. Kennelijk gelooft A dat B bij machte is een bepaalde actie uit te voeren. Volgens het model is er dan sprake van een *capability,* of *mogelijkheid* van B voor een bepaalde actie.
Het geloven in *capabilities* door de instructeur bij de uitvoerder is cruciaal voor een goed lopend dialoog. De instructeur zou anders voortdurend uitleg geven bij zaken die voor de uitvoerder als vanzelfsprekend zijn, wat uit kan lopen in frustraties of onnodige vertraging van het dialoog. [I]

5.2.3 Visualisatie werking assistent

Het model, dat in de vorige paragraaf beschreven is, is het uitgangspunt geweest voor de werking van de 2APL-agent. In het diagram hieronder is de werking van de assistent, die met de gebruiker communiceert, gemodelleerd:



Hierbij is *A* de assistent en *B* de gebruiker.
Bij het starten van de assistent, dat is na het selecteren van een MSAgent karakter en een recept, is de assistent in een wachtende toestand. Wanneer de gebruiker het gekozen recept kent, dat is *B::capable(recipe)*, doet de assistent niets totdat de gebruiker op eigen initiatief een actie uitvoert. Anders zal de assistent een instructie geven, waarop de gebruiker met een actie kan reageren.
Wanneer vervolgens een verkeerde actie gedaan is, wordt deze fout door de assistent behandeld. Bij een kleine fout, zoals het toevoegen van een verkeerd ingrediënt, kan de fout gerepareerd worden. Wanneer echter de gebruiker de gehele pan met inhoud leegt, dient het recept opnieuw gemaakt te worden.

Deze cyclus van instructies, acties en fouten afhandelen herhaalt zich totdat het recept klaar is. Wanneer het recept klaar is, gelooft de assistent dat de gebruiker in het vervolg weet hoe het recept gemaakt moet worden.

5.3 Mogelijke uitbreidingen

De Virtual Recipe Assistent is erg vatbaar voor uitbreidingen. De paragrafen hieronder zijn voorbeelden van mogelijke uitbreidingen voor alle opgeleverde producten.

5.3.1 De factor tijd in virtuele recepten

Zoals te lezen is in hoofdstuk 4.1 wordt bij een Virtual Recipe geen rekening gehouden met de factor *tijd*. Dit is wellicht de meest voor de hand liggende uitbreiding, omdat bij recepten in het dagelijks leven de kok vaak een bepaalde tijd moet wachten tot iets gaar is. Daarnaast zou dit geïmplementeerd zijn ware er genoeg tijd voor.

Hieronder bevindt zich een stukje XML-code met een voorbeeld van hoe de factor tijd in de Virtual Recipe geïmplementeerd had kunnen worden:

Voorbeeld gebruik van zelf te bepalen tools *XML code*

```
<recipe name='example'>
  <actions>
    <wait units='minutes'>
      5
    </wait>
  </actions>
</recipe>
```

5.3.2 Parallelle recept-paden

We definiëren het volgende:

SA	verzameling van acties
$E(x)$ x	wordt uitgevoerd

In de opgeleverde Virtual Recipe is een recept gedefinieerd als een reeks van acties: $A[a_1, \ldots, a_n]$ waarbij het recept dient te worden uitgevoerd op de volgende manier:

$\forall\, a \in A(E(a))$: alle acties van het recept worden achtereenvolgend uitgevoerd.

Wanneer we de Virtual Recipe echter uitbreiden met de functionaliteit dat acties parallel aan elkaar uitgevoerd kunnen worden, zijn de elementen van de reeks geen losse acties, maar verzamelingen: $LA[SA_1, \ldots, SA_n]$ en wordt een Virtual Recipe op de volgende manier uitgevoerd:

$\forall\, SA \in LA(\forall\, a \in SA(E(a)))$

Het zou dan bijvoorbeeld niet uitmaken of de gebruiker eerst water in de pan gooit en vervolgens de macaroni of andersom. In XML zou dit kunnen worden opgenomen door de acties te voorzien van ID's, waarbij acties met dezelfde ID's tegelijkertijd mogen gebeuren:

Voorbeeld parallelle acties *XML code*

```
  <insert path='0'>
    <medium_pan />
    <water amount='1000' />
  </insert>
  <insert path='0'>
    <medium_pan />
    <macaroni amount='100'>
  <insert>
```

5.3.3 De assistent vragen acties uit te voeren

Een andere mogelijke uitbreiding zou kunnen zijn dat de gebruiker de assistent vraagt om een bepaalde actie zelf uit te voeren, bijvoorbeeld omdat de gebruiker "het niet snapt". De 2APL-agent zou dan ook de bediening over de GUI moeten krijgen om aldus de gevraagde handeling uit te voeren. Als klap op de vuurpijl zou dan wellicht het gebruik van de muis door de agent gesimuleerd kunnen worden, waardoor het net lijkt alsof de agent zelf met dezelfde middelen de actie uitvoert. Java biedt hiertoe een uitstekende potentie met de klasse *java.awt.Robot*.

5.3.4 Agent-vriendelijkheid-meter

In de opgeleverde demo zal de assistent altijd vriendelijk blijven, ongeacht de hoeveelheid fouten die de gebruiker maakt en de ernst daarvan. Een mogelijke uitbreiding zou kunnen zijn dat de assistent een bepaalde graad van vriendelijkheid heeft die toe- of afneemt naarmate het recept vordert. Een leuke bijkomstigheid zou dan zijn dat ook de uit te spreken zinnen zich hierop aanpassen. Een zeer gefrustreerde assistent vertrekt natuurlijk wanneer de laatste druppel de emmer doet over lopen.

5.3.5 Het zoeken naar recepten

Deze mogelijke uitbreiding is mede door Nieske Vergunst bedacht en houdt in dat het uitkiezen van een recept een veel complexer gebeuren wordt dan het momenteel is. De gebruiker zou bijvoorbeeld moeten kunnen kiezen wat het recept in ieder geval moet bevatten, bijvoorbeeld tomaten. Of de gebruiker kan expliciet zoeken naar vegetarische recepten. Misschien dat elk recept in een bepaalde categorie geplaatst zou kunnen worden, zoals macaroni en spaghetti beiden pasta's zijn. Wellicht kan de assistent ook ingezet worden bij het kiezen van een te maken recept.

5.3.6 Voorraad

Op dit moment is er geen sprake van een voorraad, maar kan de gebruiker naar wens een oneindige hoeveelheid van ingrediënten gebruiken. Echter, in het dagelijks leven kunnen ingrediënten ook opraken. De uitbreiding zou kunnen zijn dat in een apart XML-bestand de voorraden van de ingrediënten gedefinieerd worden en dat deze hoeveelheden slinken naarmate de gebruiker de ingrediënten gebruikt. Wellicht is het hierbij noodzakelijk dat de gebruiker en de assistent kunnen improviseren met de wel aanwezige voorraad. Het zou dan bijvoorbeeld handig zijn als gedefinieerd is dat een rode paprika eventueel dienst kan doen wanneer er geen gele paprika's meer zijn.

5.3.7 Dynamisch ingrediënten maken

Een wellicht interessante, doch complexe uitbreiding zou kunnen zijn dat de gebruiker in staat is met het gegeven gereedschap de ingrediënten dusdanig te bewerken dat er sprake is van nieuwe ingrediënten. Deze ingrediënten zijn dan dynamisch aangemaakt. Denk hierbij aan het snijden van aardappels, waarna we aardappelscijfjes krijgen.

5.3.8 Huidige staat opslaan en laden

Met voorgaande uitbreidingen wordt het wellicht handig om een functie te maken waarbij de huidige staat van de environment en de assistent opgeslagen kan worden. We zouden dan bijvoorbeeld de assistent kunnen irriteren, zodat een volgende gebruiker met een chagrijnige agent te maken heeft. Of we zouden later de aardappelschijfjes kunnen gebruiken die we eerder al hebben gesneden.

6 Technische werking

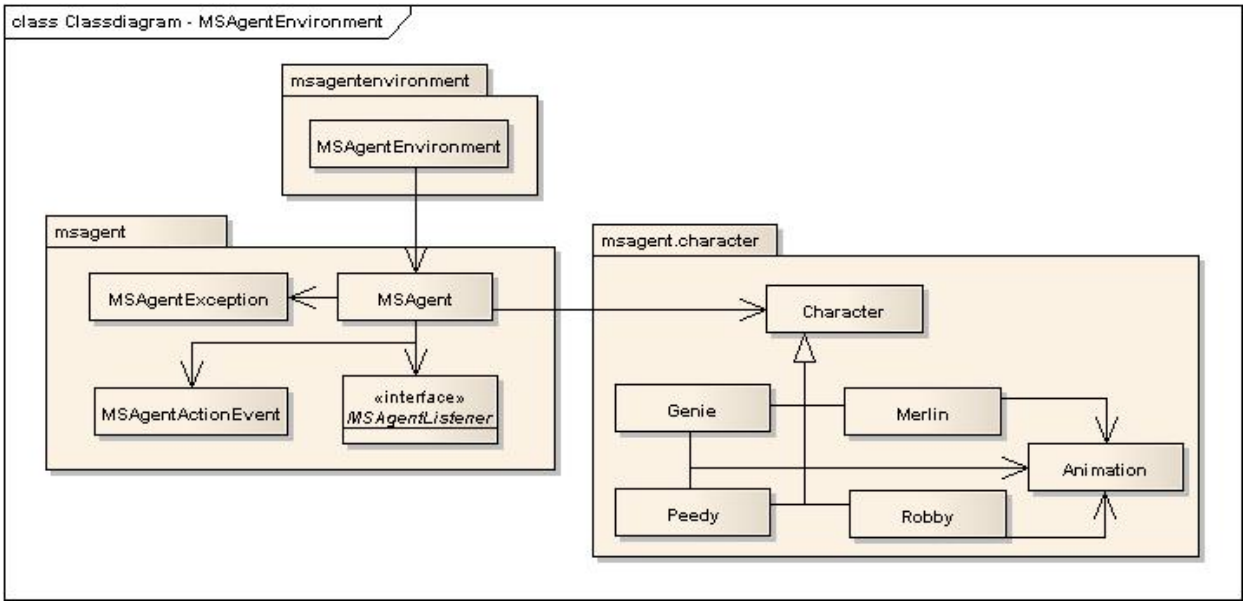
De tot nu toe besproken hoofdstukken betroffen de beschrijving van de opgeleverde producten en de motivatie per onderdeel. Dit hoofdstuk schetst een algemeen beeld van de technische opbouw van de opgeleverde producten en enige complexere of afwijkende onderdelen die extra uitleg of motivatie behoeven. De lezer wordt geacht enige kennis van *UML* te hebben, daar op sommige plaatsen UML-diagrammen gebruikt zijn om bepaalde zaken nader toe te lichten.

6.1 MSAgentEnvironment

De MSAgentEnvironment is een vrij kleine applicatie met slechts drie packages:

- **msagent;**
  Deze package bevat alle klassen die met de besturing van de Microsoft Agents te maken hebben.
- **msagent.characters;**
  Hierin zijn de klassen geplaatst die te maken hebben met de karakters van de Microsoft Agents.
- **msagentenvironment.**
  Deze package bevat slechts de main-class van de environment.

Hieronder staat het klassendiagram van de MSAgentEnvironment:



6.1.1 De karakters

Zoals in het klassendiagram te zien is, zijn de vier MSAgent-karakters gerepresenteerd door vier klassen met als superklasse: *Character.* Deze klassen zijn alle vier *singleton*[10] en hebben elk een verzameling van publieke attributen van het type *Animation.* Dit zijn de animaties die de specifieke MSAgent-karakters uit kunnen voeren.

10 Wanneer een klasse singleton gedefinieerd is, is er slechts één instantie van deze klasse.

6.1.2 De MSAgent-klasse

De MSAgent-klasse is verantwoordelijk voor de besturing van de Microsoft Agents. Er kunnen meerdere objecten van de *MSAgent* geïnstantieerd worden, zolang de karakters maar uniek blijven. Er kan dus maar één MSAgent-object gemaakt worden met bijvoorbeeld het karakter Peedy. Wanneer echter geprobeerd wordt om alsnog meerdere objecten te maken met hetzelfde karakter, verschijnt er een *MSAgentException*.

Dit gebruik van het Exception-mechanisme van Java is ingevoerd, omdat de MSAgent anders een fatale error veroorzaakt. Zoals in de volgende paragraaf te zien is, moet een MSAgent-karakter uniek zijn.

Aan een *MSAgent-object* kunnen listeners verbonden worden. Deze listeners krijgen een *MSAgentActionEvent* wanneer deze MSAgent een bepaalde actie, bijvoorbeeld praten, verricht heeft.

Voorbeeld van hoe een MSAgent-object gemaakt kan worden *Java code*

```
01  try
02  {
03    MSAgent myMSAgent = new MSAgent( Peedy.getInstance() );
04  }
05  catch (MSAgentException msae) { msae.printStackTrace(); }
```

6.1.3 De MSAgent aansturen

De MSAgent Technology is een onafhankelijke technologie van Microsoft en behoeft dus een speciale aansturing. Dit wordt bewerkstelligd door middel van een *Dynamic Linked Library* (*DLL*). Deze DLL wordt met de MSAgentEnvironment meegeleverd, zodat de MSAgentEnvironment hiervan methoden kan aanroepen.[11]

Het aanroepen van een DLL vanuit Java kan met de zogenaamde *JNI*. JNI staat voor *Java Native Interface* en is een techniek die gebruikt kan worden om OS-specifieke programma's uit te voeren. Dit gebeurt door het betreffende DLL-bestand in te laden. Dit kan worden gedaan met behulp van de Java methode $system.load(p)$, waarbij p het volledige pad naar het bestand is. Daarnaast moeten de beschikbare methoden als attributen in de klasse aangegeven worden, dit gebeurt met het keyword *native*. Deze native methoden moeten de vorm hebben zoals in de DLL beschreven staat. De native methode kan vervolgens gewoon aangeroepen worden als ware het een normale Java methode.[12]

Voorbeeld van een native methode *Java code*

```
01  private static native long SpeakAudio(int zAvatarID, String sText);
```

Voor de aansturing van de MSAgents gelden de volgende regels:

De initialisatie en besturing van de Microsoft Agents kan slechts door één thread gedaan worden.

De methoden van de Microsoft Agents blokkeren. Het terugkrijgen van de controle kan een tijdje duren, omdat animaties al gauw enkele seconden duren, net als het uitspreken van tekst.

De MSAgent-karakters zijn uniek; er mag van elk karakter slechts één instantie gemaakt worden.

Bij overtreding van de laatste regel van hierboven, zal een fatale error verschijnen. Een dergelijke fatale error is funest voor een programma, aangezien het niet opgevangen kan worden zoals dat bij Exceptions kan. Om deze reden is gepoogd alle mogelijk te verschijnen fatale errors in Exceptions op te vangen.

11 Hierbij moet gezegd worden dat deze DLL niet door mij geschreven is, maar dat Mehdi mij deze leverde.

12 Voor een meer volledige uitleg van JNI kan het document "*Een beknopte inleiding tot JNI*", dat is bijlage E, worden geraadpleegd.

6.1.4 Uitvoeren van MSAgent acties

Onder een actie van een MSAgent verstaan we elk gebruik van enige functionaliteit van de MSAgent, dus: praten, animaties uitvoeren, denken, verplaatsen, wijzen, tonen en verstoppen, maar ook het aanmaken of verwijderen van een MSAgent. Kortom: een actie kan alles zijn dat betrekking op de MSAgents heeft. Het uitvoeren van MSAgent acties wordt gedaan binnen een private klasse van de MSAgent-klasse. Deze klasse heeft een eigen *asynchrone interne thread*[13].

Tijdens het uitvoeren van een actie kan het zijn dat een andere actie ook moeten worden uitgevoerd. De MSAgent aansturing maakt hiertoe gebruik van een *queue* om deze acties achter elkaar uit te voeren.
We definiëren Q als een queue van acties $[a_0, \ldots, a_n]$. Een zo'n actie bestaat uit: een nummer om het type actie aan te duiden, een referentie naar het uitvoerende MSAgent-object om de gekoppelde listeners op te vragen en een reeks van parameters die bij de betreffende actie horen. De queue heeft twee methoden: *push:* $Q+a=[a_0, \ldots, a_n, a_{n+1}]$ en *pop:* $Q-a_0=[a_{0+1}, \ldots, a_n]$ waarbij het een voorwaarde is dat $n>0$.

We definiëren vervolgens nog de aanvullende predicaten:

$msctrl$	de thread waarbinnen de MSAgent bestuurt wordt
$S(x)$	x is suspended
$E(x)$	x wordt uitgevoerd
$R(x, y)$	x wordt uit de lijst y verwijderd
d	de actie "*detach*"

Hieronder staat de werking van de MSAgent aansturing:
$$\exists a \in Q \Rightarrow \neg S(msctrl) \wedge \forall a (E(a) \wedge R(a, Q))$$
Slechts wanneer er een actie in de queue staat, is de MSAgentControlThread actief. Dan worden alle acties uit de queue uitgevoerd en direct verwijderd. Een uitzondering op deze regel vinden we bij de actie *detach*, waarmee de MSAgent ontbonden wordt:
$$(Q+d) \rightarrow \forall a \in Q (R(a, Q) \wedge E(d))$$
Wanneer de *detach* actie aan de queue toegevoegd wordt, worden alle acties uit de queue verwijderd en wordt de MSAgent ontbonden.

Wellicht rijst nu de vraag op waarom er gekozen is om de aansturing van de MSAgents binnen een afzonderlijke thread te doen.
In paragraaf 6.1.3 hebben we kunnen zien dat de besturing van de MSAgents door slechts één thread gedaan kan worden. Met het gegeven dat elke 2APL-agent een eigen thread heeft, kunnen we dit als noodzakelijk bestempelen. Immers, we weten dat wanneer een 2APL-agent een methode van de environment aanroept, de inhoud van de methode binnen de thread van deze 2APL-agent uitgevoerd wordt. Dit zou resulteren in een conflict met deze regel.

13 Een asynchrone interne thread is per standaard ge-suspend en wordt ge-resumed op het moment dat een actie wordt aangevraagd. Na het uitvoeren van de actie is de thread wederom suspended.

6.1.5 Synchronisatie 2APL-agent en MSAgent

We hebben in voorgaande paragraaf kunnen lezen hoe het uitvoeren van MSAgent acties gedaan wordt: een 2APL-agent roept een methode aan, welke verder in een andere thread uitgevoerd wordt. Het probleem dat hier ontstaat is dat de 2APL-agent dramatisch voor gaat lopen op de MSAgent. De MSAgent voert namelijk acties uit in termen van seconden, tegenover nanoseconden van de 2APL-agent.

Hiertoe zijn de methoden van de MSAgent-klasse blokkerend gemaakt. Bij het aanroepen van een methode van de MSAgent-klasse wordt een vlag gehesen en zal de MSAgent-klasse blijven wachten in een loop. Zodra de MSAgent-klasse een MSAgentActionEvent ontvangt, dat is dus wanneer de MSAgent klaar is met een actie, wordt deze vlag neergehaald en wordt de controle van de methode ge-returned. Wanneer nu dus een 2APL-agent een Methode aanroept van de environment zal deze pas verder gaan met redeneren zodra de methode klaar is.
Het nadeel van deze aanpak is dat de 2APL-interface *bevriest* zolang de methode de controle nog niet heeft ge-returned. Dit heeft te maken met het feit dat de thread, waarbinnen de 2APL-interpreter zich bevindt, afhankelijk is van de threads van de 2APL-agents. Een alternatieve aanpak had hierom kunnen wezen dat de 2APL-agent de vlag bijhoudt en een event binnenkrijgt wanneer de actie uitgevoerd is.

Er is echter gekozen om de methode blokkerend te maken, omdat hiermee interne structuur van de 2APL-agenten ongewijzigd kan blijven. De ingreep die gemaakt zou moeten worden om de 2APL-agent synchroon te laten lopen met de MSAgent bij niet-blokkerende methoden is te groot bevonden.

6.1.6 EISification

EISification betekent zoveel als: het aanpassen van een environment zodat deze voldoet aan de eisen die gesteld zijn voor de Environment Interface Standard.

Er zijn drie belangrijke eisen waaraan voldaan moet worden:

- De EIS-klassen moeten bij de environment gevoegd worden;
- De main-klasse moet de interface *EnvironmentInterfaceStandard* implementeren of erven van de *EIDefaultImpl* klasse;
- De environment moet in een runnable-jar geplaatst worden, waarbij de main-klasse gespecificeerd is in het manifest-bestand;

Wanneer aan deze eisen voldaan worden, zou elk agent-platform, compatibel met EIS, de environment moeten kunnen gebruiken.
Daarnaast moet bij het EISifyen van een environment rekening gehouden worden met agent-platformonafhankelijkheid. Sommige agent-platformen hebben eigen klassen of typen die voor verschillende doeleinden gebruikt kunnen worden. Voor een EIS-environment kunnen deze niet gebruikt worden, tenzij ze meegenomen worden in het jar-bestand. Doch, de beste manier voor het schrijven van een EIS-environment is door in het geheel niet te kijken naar de te gebruiken agentprogammeertaal en zodoende dus volledig onafhankelijk de environment te schrijven.

6.2 VRAEnvironment

De Virtual Recipe Assistent Environment, of afgekort de VRAEnvironment, is de Java-applicatie die de environment van het agent-systeem vormt. Deze applicatie leest de virtuele recepten uit, bewaart de uitgelezen data en heeft een grafische user interface die de gebruiker bedient bij het koken van een virtueel recept.

De VRAEnvironment bestaat uit vier packages:

- **data;**
  Bevat alle klassen die de data bevatten. Deze data hebben met name betrekking op de Virtual Recipes.
- **gui;**
  Bevat alle klassen die gebruikt worden voor de visuele aspecten van de VRAEnvironment en de user-input hierbij.
- **io;**
  Bevat de klassen die gebruikt worden om de XML-bestanden uit te lezen en de afbeeldingen in te laden.
- **virtualRecipeAssistentEnvironment.**
  Deze package bevat de Main-klasse: de *VirtualRecipeAssistantEnvironment.java*.

6.2.1 Virtual Recipe in Java

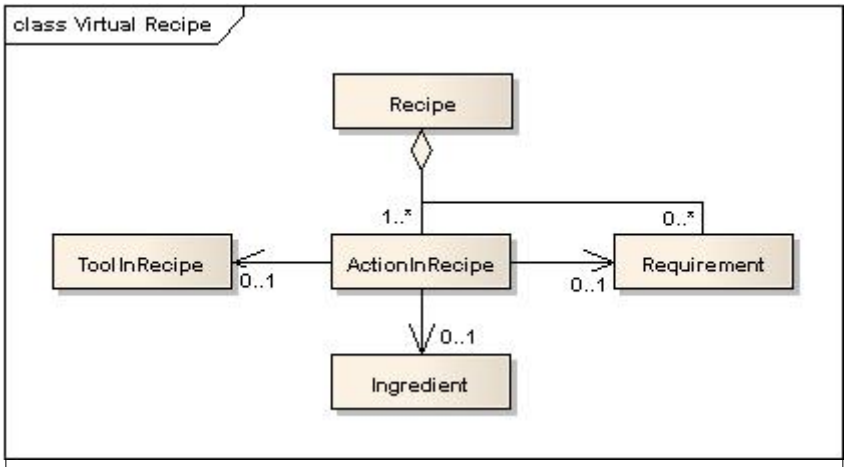
Het inlezen van de virtuele recepten wordt door de klasse *io.XMLReader* gedaan. Deze klasse maakt gebruik van de *org.w3c.dom.Document* en aanverwante klassen om de XML-bestanden uit te lezen. Allereerst worden de Tools en Ingrediënts uitgelezen, zodat bij het inlezen van de Virtual Recipies direct kan worden gecontroleerd of de gebruikte Tools en Ingredients wel bestaan. Bij een foutieve XML wordt een XMLException ge-throwed.

De klassen die betrekking hebben op de Virtual Recipies zijn de volgende:

- **Tool.java;**
- **Action.java;**
- **Ingredient.java;**
- **Recipe.java;**
- **ActionInRecipe.java;**
- **ToolInRecipe.java.**

De klassen *Tool*, *Action* en *Ingredient* worden uit de XML-bestanden *tools.xml* en *ingredients.xml* gelezen. Deze klassen vormen de definities van de tools, actions en ingrediënts. Een Tool-object bijvoorbeeld bevat een naam, titel, capaciteit en de mogelijk uit te voeren acties met de tool. Kortom: alle parameters zoals beschreven in hoofdstuk 4.1.2 .

ActionInRecipe- en ToolInRecipe-objecten zijn de actions en tools die uit de instructies van een Virtual Recipe gelezen zijn. In tegenstelling tot de klassen Tool en Action, wordt hier geen definitie gegeven van het gereedschap en de actie zelf, maar een definitie van het *gebruik* hiervan. Wanneer bijvoorbeeld in het recept wordt aangegeven dat elke willekeurige tool gebruikt kan worden om te roeren, dan zal het betreffende ToolInRecipe-object geen naam hebben, maar gekoppeld zijn aan de Action met de naam "stir".
Als bij het inlezen van de recepten tools, acties of ingrediënten gebruikt worden die niet gedefinieerd zijn, dan zal hiervoor een *XMLException* gethrowed worden, er is dan immers sprake van een incorrect XML-bestand.



Objectendiagram van de Virtual Recipe

In het objectendiagram hierboven is weergegeven hoe de structuur van de Virtual Recipe in elkaar zit. De klasse *Recipe* bestaat uit *ActionInRecipe*- en *Requirement*-objecten. We zien dat een ActionInRecipe drie gerelateerde objecten heeft: *ToolInRecipe*, *Ingrediënt* en *Requirement*. Hiermee kunnen de verschillende uitvoerbare typen acties gevormd worden[14].

Bijv.

Stel dat uit een virtueel recept de volgende actie is gelezen: $stir(tool)$. Van deze perform-action is niet specifiek aangegeven met welke tool dit gedaan moet worden. Het gegenereerde ActionInRecipe-object heeft dan geen relatie met een ToolInRecipe-object. Wanneer we in het recept $stir(spoon)$ definiëren, zal de ActionInRecipe wèl een ToolInRecipe-object hebben, namelijk met de naam "spoon".

14 Zie 4.1.1 .

6.3 De 2APL-agent

De 2APL-agent geeft de instructies aan de gebruiker voor het koken van een virtueel recept. Hierbij wordt de spraaktechnologie van de MSAgent toegepast, zodat de instructie uitgesproken wordt.

De assistent is opgebouwd uit meerdere bestanden met 2APL-broncode:

- **assistent.2apl;**
  Hier staan de initiële plannen en de PG-rules gespecificeerd. Vanuit dit bestand worden de andere bestanden ge-include.
- **assistent_beliefupdates.2apl;**
  Bevat de beliefupdates van de agent.
- **assistent_events.2apl;**
  Bevat alle PC-rules van de agent die te maken hebben met events.
- **assistent_initialbeliefs.2apl;**
  Bevat de initiële beliefs van de agent.
- **assistent_pcrules.2apl;**
  Bevat de overige PC-rules; de PC-rules die gebruikt worden als functies.
- **assistent_vocabulary.2apl.**
  Bevat de taalkennis van de agent; de verschillende zinnen die gebruikt kunnen worden.

De functionaliteiten van de agent zijn uitgesmeerd over zes verschillende bestanden, omdat de overzichtelijkheid anders verloren gaat. Elk bestand telt gemiddeld al zo'n 100 regels.

6.3.1 Het geven van de instructies

We definiëren onderstaand woordenboek:

I	de lijst met alle instructies voor een virtueel recept
s	de huidige stap in het recept
$G(x)$	de instructie x is gegeven
$D(x)$	de handeling van instructie x is uitgevoerd door de gebruiker

De algemene werking van het geven van instructies is als volgt:

$$\forall i_s \in I(\neg G(i_s) \wedge \neg D(i_s)) \rightarrow G(i_s)$$

Alle instructies, die nog niet gegeven zijn en waarvan de handelingen nog niet gedaan zijn door de gebruiker, worden gegeven. Hierbij moet gezegd worden dat de instructie i_{s+1} pas gegeven wordt, als $D(i_s)$; de agent wacht dus met het geven van een volgende instructie totdat de gebruiker de huidige actie uitgevoerd heeft. Met deze constructie kan een instructie i_s gemakkelijk herhaald worden door slechts de belief $G(i_s)$ uit de beliefbase te verwijderen.

De opbouw van de PG-rule die hiervoor gebruikt wordt is als volgt:

PG-rule voor het geven van een Perform-action *2APL code*

```
01  doAssist( yes ) <- currentStep( S ) and instruction( S, A, T, Com ) and not A = select |
02  {
03    ...
04    B( tool( T, Tn, Tc ) );
05    say( [ 'Please ', A, ' with a ', Tn, '.' ] );
06    ...
07  }
```

De goal $doAssist(yes)$ wordt geadopteerd zodra de gebruiker een recept heeft geselecteerd. In de pre-condities wordt de huidige stap opgevraagd, waarna ook de instructie die bij deze stap hoort opgevraagd wordt (indien deze bestaat). We zien dat voor deze instructie vier parameters gelden: S de stap, A de actie, T de tool en het commentaar bij de instructie.
Omdat de structuur van een PG-rule voor een Select-action gelijk is bovenstaand, moet gecontroleerd of we wel met een Perform-action te maken hebben[15]. Dit is opgelost met de regel $not\ A = select$.
Op regel 4 wordt de representatieve naam van de tool opgehaald, waarna op regel 5 de instructie daadwerkelijk gegeven wordt. Het resultaat zal zoiets zijn als: "Please stir with a spoon" of "Please cut with a knife".

6.3.2 Evaluatie van een uitgevoerde actie

Wanneer de gebruiker een instructie heeft opgevolgd, krijgt de agent een event in de PC-rules. Immers, de gebruiker voert acties met de environment uit en de environment stuurt het event. Hieronder staat een PC-rule gedefinieerd die een event opvangt:

PC-rule voor het ontvangen van een event van een Perform-action *2APL code*

```
01  event( actionperformed( A, T ), vraEnvironment ) <- true |
02  {
03    ...
04  }
```

We zien dat er een actie uitgevoerd is met twee parameters. A is de uitgevoerde actie en T de gebruikte tool. Daarnaast zien we dat het bericht van de Virtual Recipe Assistant Environment af moet komen en dat er geen pre-condities zijn. De agent zal in deze PC-rule de gegeven instructie ophalen om te controleren of de uitgevoerde actie overeenkomt met de gestelde instructie. Wanneer dit het geval is, zal de volgende instructie gegeven worden, als deze bestaat.
Wanneer daarentegen de uitgevoerde actie niet of slechts voor een deel overeenkomt met de gegeven instructie, zal de agent een instructie geven om deze fout op te lossen:

Constatering van een foute handeling *2APL code*

```
01  // What tool was this about again? Get the required tool:
02  if B( instructionPerformed( S, R, Ar, Tr, Com ) ) then
03  {
04    if B( not A = Ar ) then
05    {
06      @msAgentEnvironment( speak( ' Why did you do that? I did not tell you to do that?' ) );
07      repeatInstruction(  )
08    }
09  }
```

In de code hierboven is te zien hoe de fout geconstateerd wordt. De gegeven instructie wordt uit de beliefs gehaald (regel 2) en er wordt gecontroleerd of de actie van de gegeven instructie overeenkomt met de uitgevoerde actie (regel 4).

15 Zouden we deze controle weglaten, dan wordt een Select-action tweemaal uitgevoerd, namelijk in de daarvoor bestemde PG-rule en in bovenstaande.

6.3.3 Vocabulaire van de assistent

De agent heeft een kleine vocabulaire ingebouwd waaruit deze kan kiezen wanneer een bepaalde uitdrukking gedaan moet worden, zodat niet voortdurend dezelfde woorden gebruikt worden. In het bestand *vocabulaire.2apl* staan deze uitspraken gedefinieerd.
Deze vocabulaire bestaat uit beliefs in de vorm $<onderwerp>(<getal>,<zin>)$, waarbij het getal per onderwerp oploopt. Wanneer de agent een uitspraak wilt doen met een bepaald onderwerp, dan wordt aan de hand van een willekeurig gekozen getal de uitspraak bepaald.

Voorbeeld van de vocabulaire *XML code*

```
01  Beliefs:
02    agree( 0, 'Okay.' ).
03    agree( 1, 'Yes.' ).
04    agree( 2, 'Yes ofcourse.' ).
05    agree( 3, 'Ofcourse.' ).
06    agree( 4, 'Certainly.' ).
07    agree( 5, 'Sure.' ).
08    agree( 6, 'Yup.' ).
```

Hierboven staan alle mogelijke uitspraken die de agent kan doen wanneer deze moet beamen. Voor het kiezen van een van bovenstaande uitspraken is de volgende PC-rule geschreven:

Voorbeeld van de vocabulaire *XML code*

```
01  PC-rules:
02    agree(  ) <- true |
03    {
04      @vraEnvironment( rand(0, 7), Res );
05      B( Res = [ random( Rnd ) ] );
06      B( agree( Rnd, Text ) );
07      say( Text )
08    }
```

Dankzij bovenstaande PC-rule kan de 2APL-agent simpelweg $agree()$ aanroepen, waarna automatisch een van de zeven uitspraken gesproken wordt.

7 Proces

Dit hoofdstuk behandelt de aanpak van het project, hier worden de projectmatige aspecten van het project behandeld. De laatste paragraaf is een soort conclusie voor dit hoofdstuk waarin de sterkte- en zwaktepunten van dit project belicht worden.

7.1 Totstandkoming opdracht

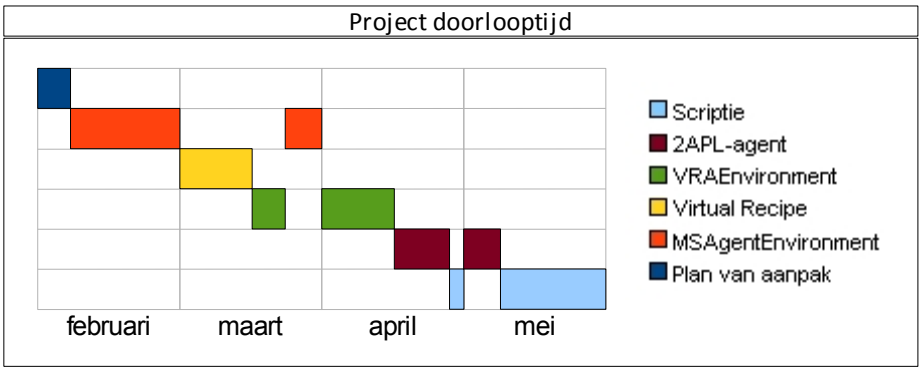
Bij mijn eerste sollicitatiegesprek met Mehdi Dastani, gedurende de maand januari, legde hij mij de keuze voor tussen verschillende opdrachten. Een van de opdrachten was het schrijven van een demo voor 2APL, waarbij Mehdi de Microsoft Agent Technologie in het bijzonder noemde. Ik heb uiteindelijk gekozen voor de opdracht om een demo te maken voor 2APL waarbij de MSAgents gebruikt worden.

Voordat de opdracht gedefinieerd werd, heeft Mehdi mij gevraagd of ik het model van een AIO[16], Nieske Vergunst, in mijn opdracht wil verwerken, hierin heb ik toegestemd. In dit model van Nieske is het assisteren van een agent bij het koken van een recept een terugkomend scenario. Daarom hebben Mehdi, Nieske en ik besloten om een demo te schrijven voor 2APL waarbij de gebruiker een virtueel recept kan koken en begeleidt wordt door een Microsoft Agent karakter. Deze opdracht is vervolgens zwart op wit gezet in het plan van aanpak.

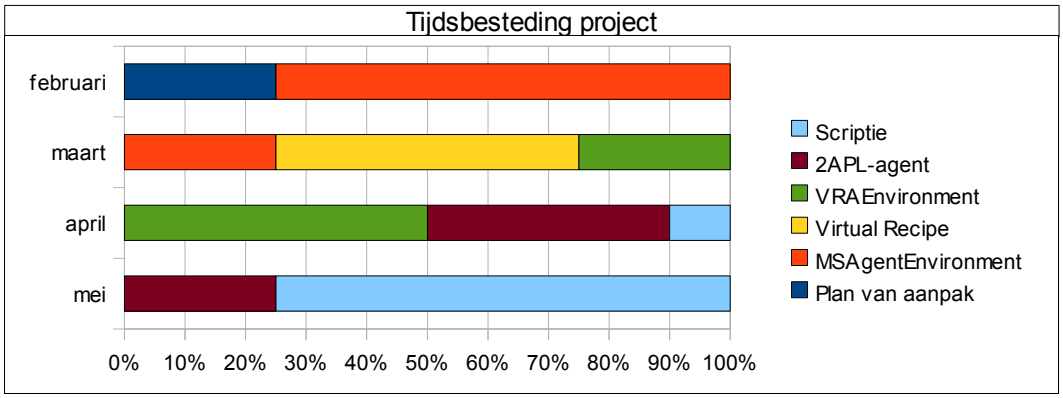
7.2 Globale aanpak

Op op 5 februari heb ik de goedkeuring van het plan van aanpak van alle partijen gekregen, dat zijn de opdrachtgevers en de begeleider. Hierna ben ik begonnen met de ontwikkeling van de MSAgentEnvironment, waarbij ik de besturing van de Microsoft Agents aan de praat probeerde te krijgen. Vervolgens heb ik mij verdiept in de Environment Interface Standard waarmee ik de MSAgentEnvironment compleet heb gemaakt. Met de oplevering van de eerste versie van de MSAgentEnvironment heb ik mij beziggehouden met het bedenken van de Virtual Recipe en hoe deze in de VRAEnvironment gebruikt zou moeten kunnen worden. Tijdens het bouwen van de VRAEnvironment heb ik de MSAgentEnvironment kunnen testen (aangezien deze hierbij gebruikt wordt) en heb ik deze nog verbeterd tot de uiteindelijke oplevering van de MSAgentEnvironment op 28 maart.
De Virtual Recipe heb ik tegelijkertijd bedacht met de VRAEnvironment, omdat deze goed op elkaar afgestemd moesten worden. Hetzelfde geldt voor de VRAEnvironment en de 2APL-agent. Ik heb de maanden april en mei afgesloten met het werken aan mijn scriptie.

In het diagram hieronder is deze globale aanpak samengevat:



16 Assistent(e) in opleiding.



In het diagram hierboven is de bezetting in tijd in procenten per onderdeel per maand geïllustreerd. Hieruit is goed te zien hoe de verschillende onderdelen elkaar overlappen.

7.2.1 Gebruikte methodieken

Zoals uit voorgaande diagrammen af te leiden is, sloot de uitvoering van de ontwikkeling van een bepaald product de ontwikkeling van een ander product niet uit; de ontwikkeling van de deelproducten hebben elkaar overlapt. Dit komt overeen met de *Watervalmethode*, waarbij de verschillende fasen in elkaar overvloeien. Toch heeft de gebruikte aanpak het meeste weg van de *DSDM* methode, omdat bijvoorbeeld de opdrachtgever nauw betrokken is geweest en de deelproducten los zijn opgeleverd. Daarnaast is in het plan van aanpak een *MoSCoW*[17] gespecificeerd.

Er is gekozen om de grote richtlijnen van de DSDM methode toe te passen, omdat het project gemakkelijk in te delen was in deelproducten. Door deze deelproducten voortijdig op te leveren krijgt de opdrachtgever een positievere indruk. Bovendien kon de opdrachtgever door praktische omstandigheden zeer gemakkelijk benaderd worden, waardoor deze door het gehele project betrokken kon blijven. Echter, van een volledige toepassing van DSDM is geen sprake, omdat er geen uitvoerend projectteam was, maar een individu.

7.3 Relatie opdrachtnemer - opdrachtgever

Zoals in het plans van aanpak beschreven staat bestaat de opdrachtgevende partij uit twee personen, namelijk: Mehdi Dastani en Nieske Vergunst. Deze opdrachtgevers zijn gedurende het gehele ontwikkelingsproces betrokken geweest bij de totstandkoming van de producten.
Met elke grote mijlpaal zijn de opdrachtgevers ingelicht om zodoende het beeld dat zij hebben van het project helder te houden. Daarnaast is het belangrijk om zo snel mogelijk feedback te krijgen over het verrichtte werk, zodat dit eventueel nog aangepast kan worden en geen problemen gaat veroorzaken in een later stadium.

De communicatie tussen de opdrachtgevers en de opdrachtnemer is zeer soepel verlopen. Zowel Mehdi als Nieske was altijd per e-mail bereikbaar en hun kantoren stonden altijd voor mij en mijn vragen/opmerkingen open.

17 Afkorting van *Must have, Should have, Could have, Won't Have*, waarmee aangegeven wordt welke prioriteiten aan de implementatie van bepaalde functionaliteiten is gehangen.

7.4 Research

Gedurende dit project heb ik onderzoek gedaan naar hoe agentgeörienteerd programmeren werkt. Hiervoor heb ik het boek *An Introduction To MultiAgent Systems* van Michael Wooldridge gebruikt. Daarnaast heb ik documentatie op het internet gebruikt van met name universiteiten.

7.4.1 Multi Agent Programming course

Het grootste deel van de research is in de vorm van het volgen van het vak *Multi Agent Programming* gedaan. Hiervoor heb ik mij als externe student aan de Universiteit Utrecht ingeschreven, waarna ik met dezelfde rechten als "normale" studenten zowel de colleges van dit vak als de practica mocht volgen.

Het vak Multi Agent Programming (*MAP*) gaat over hoe multi-agent systemen ontwikkeld kunnen worden. Hierbij leerden we twee methodologieën: *Gaia* en *Prometheus*. Deze methodologieën beschrijven hoe een MAS ontworpen kan worden met behulp van overzichten en modellen. Daarnaast maakten we kennis met enkele agentgeörienteerde programmeertalen: *Jason*, *Jadex* en *2APL*. We moesten in de practica een online marktplaats ontwerpen, waarbij twee agenten een verkoopproces afhandelen.

De colleges werden elke woensdag gegeven en duurde twee uur. Het practicum was 's vrijdags, officieel ook twee uur, maar dit was niet verplicht.

7.4.2 Conclusie

Het vak dat ik heb gevolgd was helder en ook interessant voor het project. Hoewel ik al wel eerder kennis van 2APL had opgedaan in een voorgaand onderzoekssemester, heeft dit vak mij meer duidelijkheid gegeven in de theorie achter 2APL. Bovendien heb ik er een beter beeld van Multi Agent Systemen door gekregen.

7.5 Sterkte- en zwaktepunten

De aanpak van dit project kent sterkte- en zwaktepunten. Deze zijn hieronder beschreven.

7.5.1 Opdracht formulering

De opdrachtformulering is zeer soepel verlopen. De opdrachtgevers waren erg helder over de eisen die zij stelden aan het product. Daarnaast had ik al enige kennis van de te gebruiken technieken waardoor een inschatting van de benodigde tijd voor de opdracht direct kon worden gedaan zonder voorafgaande research. Mede hierdoor heb ik in een vroeg stadium van het project het plan van aanpak op kunnen leveren.

7.5.2 Inschatting Virtual Recipe

Het uitwerken van de Virtual Recipe heeft meer tijd gekost dan nodig. Dit heeft te maken met een slechte inschatting aan werk dat de Virtual Recipe levert. Ik heb er te weinig rekening mee gehouden dat een Virtual Recipe in drie verschillen talen moest worden gedefinieerd en geconverteerd moest worden. Hierdoor heb ik in het begin de lat van een Virtual Recipe te hoog gelegd. Op het moment dat ik constateerde dat de gestelde mogelijkheden voor de Virtual Recipe de haalbaarheid van het project in gevaar zouden kunnen brengen heb ik de grenzen van de Virtual Recipe moeten bijstellen.

8 Conclusie

Met dit hoofdstuk wordt dit document afgesloten. In voorgaande hoofdstukken is de werking van de opgeleverde producten beschreven en de technische bouw hiervan. Dit hoofdstuk dient voornamelijk als een reflectie op het geheel van het project en de doelstelling in het bijzonder.

8.1 Aanbevelingen

Bij het uitvoeren van een project waarbij Agent Technology toegepast wordt is het aan te raden hier van tevoren enige kennis van op te doen, omdat het onderwerp te breed is om binnen een half jaar voldoende eigen te maken en het op een juiste manier toe te passen. Helaas biedt internet op dit moment onvoldoende informatie hiertoe. Een interessant boek om te overwegen over dit onderwerp is *An Introduction to Multi Agent Systems*, door Michael Wooldridge.

Daarnaast is de programmeertaal 2APL dusdanig afwijkend van andere, wat bekendere, programmeertalen dat ook hier beter vooraf in verdiept kan worden. Agentgeörienteerd programmeren is een geheel andere manier van programmeren dat enig geduld en toewijding nodig heeft om het onder de knie te krijgen.

8.2 Reflectie op de doelstelling

In het plan van aanpak en de inleiding van dit document was de doelstelling als volgt geformuleerd:

- Het ontwikkelen van een aansprekende demo voor 2APL, waarbij het model *Agent-Based Speech Act Generation* toegepast wordt en welke compatibel is met de *EIS standaard*.

Van de opgeleverde producten is de Virtual Recipe Assistant het eindproduct. Hierbij moest men een virtueel recept koken met behulp van een Microsoft Agent cartoon-figuur. Dit is mijns inziens een redelijk aansprekende demo, omdat het koken een leuke interactie tussen de gebruiker en de agent biedt en de Microsoft Agent met grappige animaties en tekstjes op de input van de gebruiker reageert. Voor de technische werking van de assistent is het model *Agent-Based Speech Act Generation* benut. Dit is zichtbaar in de afhandeling van de fouten die de gebruiker maakt bij het koken. Deze Virtual Recipe Assistant demo is tevens gebouwd volgens de specificaties van de EIS en is dus inzetbaar bij meerdere agent platforms.

Met de oplevering van de Virtual Recipe Assistant als demo is naar mijn mening aan bovenstaande doelstelling voldaan.

8.3 Reflectie op 2APL

Mijn eerste kennismaking met 2APL was ergens in de maand oktober. Zoals ook al in het Voorwoord beschreven staat, was dit gedurende een onderzoekssemester naar de Agent Technology. Voor dit onderzoek heb ik in een team van in totaal drie studenten een simulatie van de stoelendans in 2APL geprobeerd te maken. Hierin zijn we toen echter niet geslaagd met als voornaamste reden dat 2APL nog veel aan documentatie miste. De gebrekkige documentatie, gecombineerd met de (van de meest bekende programmeertalen) afwijkende syntax, heeft mijn beeld van 2APL een ietwat negatieve draai gegeven.

Echter, na meer ervaring met 2APL en enkele andere agentgeoriënteerde programmeertalen opgedaan te hebben, heb ik mijn mening hierover herzien. De logica achter de syntax van 2APL is mij, nu ik meer inzicht in de predicatenlogica heb, veel duidelijker geworden. Daarnaast is het BDI-model op een, naar mijn mening, logische en simpele manier geïmplementeerd. Dit in tegenstelling tot bijvoorbeeld Jason, waarbij, na het uitvoeren van een plan voor een gesteld doel, het doel als "voldaan" beschouwd wordt, ongeacht het resultaat van dit plan.

8.3.1 Mogelijke verbeterpunten van 2APL

Zoals ook in de Inleiding te lezen is, bevindt 2APL zich nog in de ontwikkelingsfase. Ik heb zelf de volgende verbeterpunten voor 2APL bedacht:

- **Specifiekere errors van de parser;**
  Op dit moment worden de fouten in de 2APL-bestanden heel gebrekkig gemeld. Zo wordt bijvoorbeeld een gehele PG-rule aangeduid als incorrect, terwijl niet specifiek aangegeven wordt waar binnen de PG-rule de fout zit.
- **Toevoegen van de "elseif";**
  Wanneer binnen de if-then-else-structuur voor de else ook een voorwaarde gesteld dient te worden, moet deze momenteel genest worden, met het resultaat dat de code een stuk groter en minder fraai wordt bij veel geneste if-then-else-structuren. Door het invoeren van een "*elseif*" kan een regel uitgespaard worden. Bovendien strookt dit beter met de meeste programmeertalen.
- **Standaard PC-rules invoeren;**
  Sommige functies zijn zo universeel gebruikt, dat het mij beter lijkt deze standaard in de PC-rules mee te leveren, dan dat enkele afzonderlijke environment deze functies moet schrijven. Een voorbeeld functie is de *random*, waarmee willekeurige getallen gegenereerd kunnen worden. Een ander voorbeeld is de *wait*, waarmee een timer gezet kan worden.
- **Betere documentatie.**
  Een bekend probleem van 2APL is de documentatie. Wellicht dat er een soort van API gemaakt kan worden.

8.4 Slot

Wanneer ik terugblik op de verlopen vier maanden waarin ik dit project heb mogen uitvoeren kan ik stellen dat deze vier leerzame en plezierige maanden waren. Ik vond het een erg leuk en bevredigend project waar ik veel van mijn op de HU geleerde kennis op los kon laten. Bovendien heb ik van veel vrijheid en zelfstandigheid mogen genieten bij dit project wat het erg aangenaam heeft gemaakt. Ook het volgen van de college's en het practicum van het vak Multi Agent Programming heeft mij erg aangesproken en bovendien geholpen bij dit project.

Met het succesvol afsluiten van dit project en mijn bachelorfase in het algemeen hoop ik volgend studiejaar, dat is september 2010, te beginnen met een master aan de Universiteit Utrecht. Waar ik vijf maanden geleden nog twijfelde over de te kiezen studierichting, ben ik er nu zeker van dat dit zeker Agent Technology moet gaan worden. Dit afstudeerproject heeft sterk bijgedragen aan mijn keuze hiervoor.

Woordenlijst

Literatuurlijst

[a] http://en.wikipedia.org/wiki/Object-oriented_programming
[b] Michael Wooldridge - An Introduction to Multiagent Systems 2nd edition
[c] Van Dale woordenboek
[d] http://en.wikipedia.org/wiki/Swarm_intelligence
[e] http://www.csc.liv.ac.uk/~mjw/Prof_Michael_Wooldridge_-_Home_Page/Prof_Michael_Wooldridge_-_Home_Page_2.html
[f] http://en.wikipedia.org/wiki/Belief-Desire-Intention_model
[g] 2APL A Practical Agent Programming Language - Mehdi Dastani
[h] http://msdn.microsoft.com/en-us/library/ms695784(VS.85).aspx
[i] http://en.wikipedia.org/wiki/Microsoft_Agent
[j] Environment Interface Implementation Guide for EIS v0.2 - Tristan M. Behrens
[k] http://nl.wikipedia.org/wiki/Kookkunst
[l] N.L. Vergunst - BDI-based error handling in task-oriented human-computer dialogues (2010, to be published)

BIJLAGE A

Plan van aanpak

Virtual Recipe Assistant

Document Informatie

Auteur:	Hagenaars, R
Document titel:	Plan van aanpak – Virtual Recipe Assistant
Versie:	1.0
Laatst aangepast:	2010/02/04

Inhoud

Achtergrond

Korte beschrijving van de opdrachtgever

De Universiteit Utrecht, ofwel UU , is een groot en veelzijdig kenniscentrum waar onderwijs en onderzoek geboden wordt op internationaal vlak. De UU behoort tot 's werelds topuniversiteiten en staat op nummer 1 van universiteiten in Nederland.

dr. M. Dastani, een docent aan de UU, is de persoonlijke opdrachtgever. Als docent in de Agent technologie heeft hij meegewerkt aan de totstandkoming van de programmeertaal 2APL.

Betrokken partijen

Aan dit project is het volgend aantal betrokken partijen verbonden:

- **dr. M. Dastani;**
  De primaire opdrachtgever. Is direct betrokken bij het project.
- **N. Vergunst;**
  Tweede opdrachtgever. Zij is bedenker van het model *Agent-Based Speech Act Generation*.
- **T. Behrens;**
  Contactpersoon voor de *EIS* standaard.
- **drs. L. van Moergestel;**
  Indirect betrokken partij. Ondersteunt de student bij de procedurele kwesties van de Hogeschool Utrecht.
- **R. Hagenaars.**
  Uitvoerende partij.

Huidige situatie

2APL is een relatief jonge taal waar nog maar enkele demo's voor geschreven zijn. In de officiële release is slechts 1 demo aanwezig, waarbij twee agenten (Harry en Sally) bommen zoeken en opruimen.

Aanleiding

De geringe hoeveelheid aan demo's voor 2APL drukt de concurrerende positie met andere agent programmeertalen als Jason en Jadex. Hieruit is de behoefte ontstaan naar nieuwe demo´s.
Daarnaast is het van belang om grote hoeveelheid demo's te hebben om ontwikkelaars vertrouwd te maken met de agent programmeertaal en de functies en mogelijkheden hiervan.

Opdracht

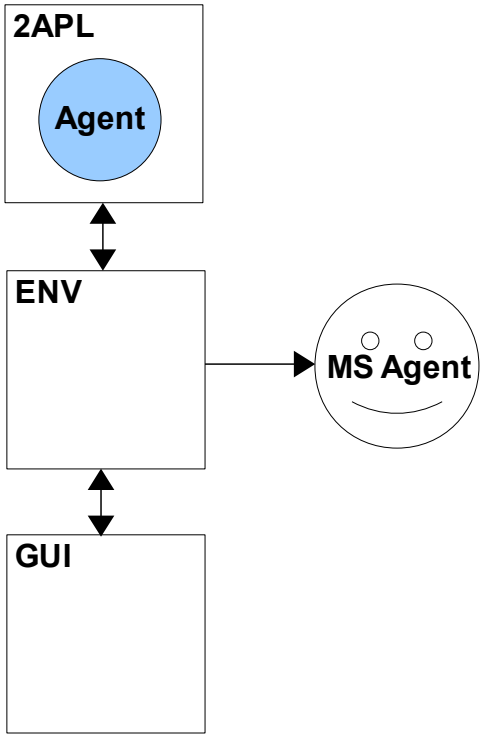
Doelstelling

Het ontwikkelen van een aansprekende demo voor 2APL, waarbij het model *Agent-Based Speech Act Generation*[1] toegepast wordt en welke compatibel is met de *EIS standaard*[2].

Omschrijving

Het project heeft de naam *Virtual Recipe Assistant* gekregen. Het idee is dat een gebruiker voor een bepaald recept de instructies van de agent opvolgt. De gebruiker kookt dus een virtueel recept, waarbij de agent toezicht houdt en aanbevelingen doet.

In figuur 1 is de globale opzet van Virtual Recipe Assistant geïllustreerd. Hieronder worden de verschillende onderdelen, zoals in de figuur hiernaast weergegeven is, beschreven:



Figuur 1. Virtual Recipe Assistant globaal

De Agent

Dit is het onderdeel van het systeem dat (kunstmatig) redeneert, oftewel het brein van het systeem. De Agent zal geschreven worden in het 2APL platform. Hierin zullen de aannames, alsmede de intenties en acties gedefinieerd worden volgens het BDI model.

De Environment

De Environment doet dienst als bemiddelaar tussen de Agent programmeertaal, 2APL in dit geval, en de "randapparatuur", te weten de GUI en de MS Agent. Deze Environment wordt geschreven in Java. Er is een standaard gespecificeerd voor het schrijven van Environments voor agent programmeertalen (zie EIS). De te schrijven Environment voor het project zal dan ook deze standaard respecteren.

1 Model voor de communicatie tussen agenten. Dit model zal als uitgangspunt worden gebruikt en is beschreven onder het kopje "Agent-Based Speech Act Generation."

2 De EIS standaard, wat staat voor Environment Interface Standard. Zie het betreffende kopje voor meer informatie.

De GUI

De GUI (Graphical User Interface) zal dienst doen als communicatiemiddel van de gebruiker naar de agent. Het zal de vorm hebben van een Window met verschillende knoppen die staan voor acties en spraak. Hiermee kan de gebruiker dus bijvoorbeeld klikken op een knop welke staat voor een specifiek ingrediënt, waarna de Agent feedback zal geven over deze actie.

De visuele weergave van de agent

Om de demo aansprekender te maken is ervoor gekozen om een visuele weergave van de agent aan het systeem te koppelen. Deze weergave zal zijn in de vorm van een *Microsoft Agent* (of *MS Agent*). Een MS Agent is een grappig cartoon achtig figuurtje die op commando tekst uit kan spreken en kleine handelingen kan uitvoeren. Bij deze handelingen moet men denken aan animaties voor bijvoorbeeld zwaaien, dansen, etc.

Figuur 2. Merlin de MS Agent

Deze MS Agent zal door de Environment aangestuurd worden. Omdat de 2APL Agent toegang heeft tot de Environment, kan deze hiermee indirect de MS Agent aansturen. Zodoende zal de MS Agent de 2APL Agent visueel

Agent-Based Speech Act Generation

Dit model, welk onder andere bedacht is door Nieske Vergunst, zal worden toegepast in de te ontwikkelen demo. Agent-Based Speech Act Generation beschrijft de communicatie tussen verschillende agenten. In dit geval de gebruiker en de 2APL agent.
In dit model wordt naar een optimale balans tussen proactief en reactief gestreefd, wat volgens dit model de gewenste vorm van communicatie van een agent is. Tussen de agenten is sprake van een gemeenschappelijk doel, wat hier het succesvol bereiden van een recept is. De 2APL agent gaat uit van bepaalde kennis bij de gebruiker, zoals hoe water aan de kook gebracht kan worden. De mate waarin de agent bereid is om de gebruiker te leren hoe basis activiteiten gedaan moeten worden zal afhangen van de tijd voor het project.

EIS

EIS staat voor Environment Interface Standard en beschrijft hoe een Environment geïmplementeerd dient te worden. Wanneer een Environment deze standaard respecteert, zal deze toegankelijk voor elke willekeurige agent programmeertaal taal worden die EIS Environments ondersteunen. 2APL is met de laatste releases een van de agent talen die compatibel is met EIS.

Projectgrenzen

Hieronder staat een aantal functionaliteiten van de op te leveren applicatie die opgenomen worden, dus binnen de projectgrenzen vallen. De waarden van deze functionaliteiten zullen worden beschreven aan de hand van het *MoSCoW model*[3]:

M	:	2APL agent die reageert op de events die binnenkomen na het drukken op de knoppen van de GUI;
M	:	Environment volgens EIS welke de MS Agent aan zal sturen;
M	:	De GUI in de vorm van een paneel met knoppen, waarbij de knoppen mogelijk veranderen naar aanleiding van eerdere handelingen of aanwijzingen van de agent;
S	:	Mogelijkheid om met XML recepten toe te voegen;
S	:	Mogelijkheid om met XML de ingrediënten van de recepten toe te voegen;
S	:	Mogelijkheid om met XML het gereedschap toe te voegen;
S	:	Mogelijkheid om met XML de handelingen toe te voegen;
C	:	Mogelijkheid om een MS Agent te kiezen;
W	:	Mogelijkheid om meerdere 2APL agenten in te zetten.

3 Voor lezers die niet bekend zijn met het MoSCow model, hier een korte uitleg. De functionaliteiten zijn voorzien van een letter waarbij geldt dat: M in het eindproduct moet voorkomen, S in het eindproduct zou moeten voorkomen, maar waarvoor een andere vorm ook mogelijk is, C in het eindproduct zal voorkomen als hiervoor genoeg tijd is, W buiten de projectgrenzen valt, maar een optie voor toekomstige uitbreiding vormt.

Producten

Hieronder staan alle op te leveren producten beschreven gerangschikt op chronologische volgorde:

Opleveringen

Verwachte datum[4]	Product
05-02-10	Plan van aanpak
01-06-10	Virtual Recipe Assistant
01-06-10	Scriptie
04-06-10	Bedrijfsbeoordeling
Nog niet bekend	Presentatie

Plan van aanpak

Hierin wordt beschreven hoe het project aangepakt wordt, de randvoorwaarden, projectgrenzen etc. Bij goedkeuring van dit document door de bedrijfsbegeleider en docentbegeleider zal de ontwikkeling van het project aanvangen.

Virtual Recipe Assistant

Dit is een werkende applicatie, geschreven in een combinatie van Java, 2APL en mogelijk XML of Visual Basic. Bij oplevering zal zowel de gecompileerde code als de broncode vrijgegeven worden. De broncode zal voorzien zijn van Javadoc.
(Zie ook hoofdstuk Opdracht - Omschrijving)

Scriptie

De scriptie zal beschrijven hoe de uitvoerende student te werk gegaan is, dus het *proces*. Daarnaast zal het documentatie bij Virtual Recipe Assistant bevatten. Verder zal worden beschreven *hoe* Virtual Recipe Assitant geschreven is, een technische handleiding.

Bedrijfsbeoordeling

Dit zal een document zijn waarin de bedrijfsbegeleider de uitvoerende kracht beoordeelt.

4 Deze datum is een streefdatum. In realiteit zou deze eventueel af kunnen wijken.

Presentatie

De presentatie vormt een korte samenvatting van de scriptie en verhaalt dus het doorlopen proces en de uiteindelijke werking van Virtual Recipe Assistant.

Aanpak

Randvoorwaarden

Om het project succesvol af te kunnen ronden zal aan een aantal randvoorwaarden moeten worden voldaan, welke hieronder geformuleerd staan:

- De uitvoerende student moet een werkplek hebben waar hij zijn producten kan ontwikkelen. De eisen aan deze werkplek zijn: er moet een computer staan of er is ruimte voor een laptop, er moet toegang zijn tot het internet, de benodigde software is aanwezig;
- De uitvoerende student moet de tijd welke per onderdeel gepland is in de planning beschikbaar krijgen;
- Voor vragen over het model Agent-Based Speech Act Generation kan de student terecht bij de partij die hiervoor verantwoordelijk is. Deze partij zal de vraag op korte termijn, afhankelijk van de spoed, beantwoorden;
- Voor verdere inhoudelijke vragen over een op te leveren product kan de student terecht bij de opdrachtgever;
- Voor begeleiding omzake proces betreffende kwesties kan de student terecht bij de docentbegeleider die ofwel bereid is per mail support te leveren of een afspraak wil maken.

Kwaliteitswaarborging

Om de kwaliteit van de eindproducten te waarborgen zal hiervoor per product feedback van een begeleider of de opdrachtgever gevraagd worden gedurende het ontwikkelingsproces.

Hieronder wordt aangegeven per product[5] wie hiervoor benaderd zal worden:

- Het plan van aanpak zal ter goedkeuring worden getoond aan alle betrokken patijen. Bij afkeuring zal deze verbeterd worden;
- De Virtual Recipe Assistant zal getoond worden aan N. Vergunst en dr. M. Dastani. Daarnaast zal deze door Tristan Behrens bekeken worden om te controleren of de Environment EIS compatibel is;
- Voor de scriptie zal feedback gevraagd worden aan drs. L. van Moergestel en dr. M. Dastani;
- De punten voor de presentatie zullen aan drs. L. van Moergestel getoond worden.

Naast het vragen van feedback aan bovengenoemde partijen, zal gestreefd worden naar een zo consequent mogelijke broncode.

Risico's

Voor het uit te voeren project zijn de volgende risico's voorzien:

- **Ziekte of absentie;**
  Wanneer een betrokken partij ziek wordt of voor langere tijd afwezig is, kan dit gevolgen hebben voor de uitvoering van het project. Afhankelijkheid van betrokken partijen zal dan ook zo veel mogelijk vermeden worden. Wanneer echter de uitvoerende student zelf ziek wordt, moeten eventueel de projectgrenzen worden aangepast.
- **Technische problemen.**
  Wanneer zich technische problemen voordoen zoals het crashen van een computer, zal dit directe gevolgen hebben. Om de gevolgen hiervan zoveel mogelijk in te perken moeten de op te leveren producten met regelmaat gebackupped worden.

5 De oplettende lezer zal opmerken dat de bedrijfsbeoordeling vermeld wordt als op te leveren product, doch hier niet tussen staat. De reden hiervoor is dat deze beoordeling niet geschreven wordt door de uitvoerende student, maar door de bedrijfsbegeleider. De kwaliteit van dit product ligt dan ook in zijn handen.

Planning

Hieronder staat de planning voor de komende vijf maanden. De eerste tabel beschrijft de verschillende actie's en koppelt deze aan een ID. De tweede tabel bevat de data en het aantal dagen per actie. Deze planning is geen absolute garantie voor de daadwerkelijke uitvoering.

Acties

Actie ID	Naam actie	Omschrijving
#1	Plan van aanpak	Hieronder valt het geheel van de opdracht formulering tot de uiteindelijke goedkeuring hiervan.
#2	Environment	Het onderzoeken van hoe de Environment gekoppeld wordt aan 2APl en de MSAgents en hoe deze EIS compatibel gemaakt kan worden.
#3	GUI	Het opstellen van een plan van hoe de GUI gemaakt zal worden. Hierbij moet men denken aan een klassendiagram. Daarnaast zal dit teruggekoppeld worden naar de opdrachtgever.
#4	2APL agenten	Het onderzoeken van de 2APL agenten met het model dat hiervoor gebruikt wordt.
#5	Ontwikkeling	Het daadwerkelijk ontwikkelen van de Virtual Agent Assistant.
#6	Scriptie concept	Het schrijven van een concept voor de scriptie. Hierbij wordt feedback gevraagd van de docentbegeleider.
#7	Scriptie	Het daadwerkelijk schrijven van de scriptie.
#8	Presentatie	Het voorbereiden voor de eindpresentatie van het gehele project.

Data en tijdbesteding

Actie ID	Startdatum	Einddatum	Dagen
#1	01-02-10	06-02-10	5
#2	08-02-10	13-02-10	5
#3	15-02-10	20-02-10	5
#4	22-02-10	27-02-10	5
#5	01-03-10	30-04-10	35
#6	01-04-10	30-04-10	5
#7	03-05-10	31-05-10	20
#8	01-06-10	Onbekend	5

Bronvermelding

Hieronder staat een lijst van bronnen die zijn gebuikt bij de totstandkoming van dit document:

- N.L. Vergunst - BDI-based error handling in task-oriented human-computer dialogues (2010, to be published);
- eisproposal.pdf (http://sourceforge.net/projects/apleis/develop);
- Afstudeervoorstel.doc (R. Hagenaars);
- Afstudeerleidraad cluster ICT vt dt du cursus 2009-2010.pdf;
- jaarrooster 2009–2010.pdf (https://www.roosters.hu.nl/0910/jaarroosters/FNT_ICT/jaarrooster%202009-2010.pdf).

BIJLAGE B

Utrecht, 23-03-2010

Environment Interface Standard Documentation
MSAgentEnvironment

Author: Hagenaars, Ramòn

Introduction

This document describes the MSAgentEnvironment as released on the 23th of march 2010, conform the Environment Interface Documentation Policy by T. M. Behrens. For this document, knowledge of the Environment Interface Standard is required since terms like *Percepts* and *Entities* will not be explained here.

The MSAgentEnvironment has a folder called *"dll"* which contains a dll-file that is necessary to run the Microsoft agents. In order to make the environment find this library, it must remain in the same directory as the *msagentenvironment.jar*. Do not rename or move the jar-file or the dll-file.

Documentation

Environment description: The MSAgentEnvironment is a simple Environment which can be used to control Microsoft Agent characters.
Please note that in order to run this environment, you must be running a Windows platform and have installed the required drivers for the agents. If the drivers are not installed yet, please read the user guide which describes where to find these drivers.

Jar-file: *msagentenvironment.jar*.

Entities: There are 4 possible entities:

- **Merlin :** a funny looking wizard, probably the most well-known MS agent;
- **Genie :** a ghost with an Arabic look;
- **Peedy :** a green parrot;
- **Robby:** a robot with a robot voice.

Each entity is unique; it is not possible to create two agents with the same entity.
The entities are of the *MSAgent* class and have one *Character* object attached with them. Each Character has a predefined set of animations, which can be found in the Javadoc.

Actions: We define the following actions and their arguments:

show (*Identifier*)	Makes the MSAgent appear on the screen. The identifier is to determine whether to execute the action animated or just plain. It should be either *"yes"* or *"no"*.
hide (*Identifier*)	Makes the MSAgent disappear. Again, the identifier determines whether to animate or not while hiding and should be "yes" or "no".
speak (*Identifier*)	Lets the MSAgent say a text. The identifier is the text to speak.
think (*Identifier*)	Lets the MSAgent think a text. The identifier is the text to think.
gestureAt (*Numeral* , *Numeral*)	Lets the MSAgent gesture at a location on the screen. The numbers represent x and y coordinates with respect to the screen.
moveTo (*Numeral* , *Numeral* , *Numeral*)	Moves the MSAgent to a specific location on the screen. If the MSAgent is visible while executing this action, it will perform this action animated. The first two numbers represent x and y coordinates with respect to the screen. The third argument determines how long it should take to move to the position.
perform (*Identifier*)	Performs a specific animation, where the identifier is the name of the animation. Please visit the MSDN website or see the Javadoc for all possible animations performed per Character.

All actions as well as their arguments are case insensitive.

Percepts: For each action you will receive a percept after the action was executed successfully. You will receive: $msaae(c)$, where c is the character of the agent which performed the action and where $msaae$ stands for *MSAgentActionEvent*, which is to indicate that a specific action was executed. For example, you could get: $show(peedy)$.

$msaae$ can be one of the following:

- createavatar;
- detach;
- gestureat;
- hide;
- initial;
- moveto;
- playanimation;
- setlanguageid;
- show;
- speakaudio;
- speaktext;
- think.

Environment-management: The environment can be initialized by specifying the MSAgents to load. So for example, if only Genie and Robby should be loaded, the set of arguments would be something like $P = genie, robby$.

Note that if no arguments are set, the environment will load all four of the MSAgents.

BIJLAGE C

Utrecht, march 22, '10

User guide to the *MSAgentEnvironment*

Author: Hagenaars, Ramòn

Introduction

This document will describe each step that should be taken in order to run the MSAgentEnvironment examples for 2APL. If you already know how to install Microsoft agents or if you already have installed them, you do not need to read this document.

Please note that it is assumed that **you already have installed the newest 2APL** and that **you are slightly familiar** with working with it. If not, please read the 2APL user guide which can be downloaded from the following URL:

- http://www.cs.uu.nl/2apl/downloads.html

Requirements

Before reading, downloading and installing, make sure you meet the following requirements:

- Windows NT 4, 2000, ME, XP or newer;
- At least 20 mb RAM;
- A 100-MHz processor or faster;
- 27 mb of free disk space

If you do meet with the requirements stated above, please continue with step 1.

Step 1: Download and install the environment

The first thing you need to do is to download the MSAgentEnvironment. It can be found on the 2APL Sourceforge website.
After downloading, please open the zip-file and extract it's content (a folder called *"msagent"*) to any location on your disk you like. Since it is an example, we suggest that you place it in the *"examples"* folder in your 2APL installation.
If you open the msagent folder, you will find another folder called *"doc"*. In there you will find some useful documentation for developing your own msagent projects.

Step 2: Download and install Microsoft agents

Microsoft Agent Core Components

If you are running Microsoft Windows ME, Windows 2000 or Windows XP, you already have an installation of these components on your computer and may thus skip this file. Otherwise, please download and install the file from the following source:

- http://activex.microsoft.com/activex/controls/agent2/MSagent.exe

Merlin, Genie, Peedy and Robby

Now that you have installed the Core Components, you will need to install the four Microsoft Agent characters.

For **Genie**, use the following URL:

- http://download.microsoft.com/download/0/0/c/00cde5f8-321d-4325-baae-eb27f1bde85f/Genie.exe

For **Peedy**:

- http://download.microsoft.com/download/a/f/5/af572f68-b83e-4e2c-8b0f-fd5fadf588e7/Peedy.exe

For **Robby**:

- http://download.microsoft.com/download/2/b/9/2b904bbb-c0b1-4840-b332-ba0615d1041e/Robby.exe

Merlin however is installed already if you are running Microsoft Windows ME, Windows 2000 or Windows XP. Otherwise, please use the following URL:

- http://download.microsoft.com/download/1/d/b/1dbee406-9b5f-48c5-b901-dd1a3f3c4669/Merlin.exe

Now that you have installed the Core Components and the four characters Merlin, Genie, Peedy and Robby, you are ready to run the MSAgentEnvironment. **However**, if you want the agents to actually speak (with audio) your text, please continue with this user guide.

Speech API (SAPI 4.0)

The next file we need is the Speech API, also known as SAPI. **Please note** that we need SAPI **4.0**, which **is not the newest version**! Luckily it is possible to have both SAPI 4.0 ánd a newer version on your Windows. Please download and install from the following URL:

- http://activex.microsoft.com/activex/controls/sapi/spchapi.exe

Learnout & Hauspie TruVoice

The last file you need to download and install is the Learnout & Hauspie TruVoice Text To speech Engine. You need the American English TTS which you can download from the following location:

- http://activex.microsoft.com/activex/controls/agent2/tv_enua.exe

Step 3: Run the examples

If you have performed step 1 and 2 you are now ready to run the examples, which are provided with the MSAgentEnvironment.
Please run 2APL and open the *"msagent.mas"* in one of the two example folders. A Microsoft Agent (or two, depending on which example you run) should now appear and speak with audio.

BIJLAGE D

Utrecht, 25-05-2010

Environment Interface Standard Documentation
VirtualRecipeAssistentEnvironment

Author: Hagenaars, Ramòn

Introduction

This document describes the VirtualRecipeAssistentEnvironment as released on the 25th of march 2010, conform the Environment Interface Documentation Policy by T. M. Behrens. For this document, knowledge of the Environment Interface Standard is required since terms like *Percepts* and *Entities* will not be explained here.

The VirtualRecipeAssistentEnvironment, or VRAEnvironment, uses the MSAgentEnvironment. This environment requires some drivers to run, so please read the "Userguide to the MSAgentEnvironment" document, which describes every step you should take in order to let the MSAgentEnvironment be able to run.

Documentation

Environment description: The VRAEnvironment enables the user to cook a virtual recipe. During the cooking process, you will be assisted by an agent represented as an MSAgent. This agent will guide you through all the steps of a certain recipe by giving instructions and feedback. These virtual recipes, as well as the required ingrediënts and tools, are defined in XML. The XML-files can be found "xml" directory.

Jar-file: *virtualrecipeassistent.jar*.

Entities: The VRAEnvironment does not define any entity.

Actions: We define the following actions and their arguments:

concat (*Identifier*$_0$, … , *Identifier*$_n$)	Concatenates two or more strings up to $n>0$.
rand (*Numeral* , *Numeral*)	Returns a random number between x en y , where $x<y$ and y is exclusive.
addSpeak (*Identifier*)	Adds a selection property to the selectionboix with which the user can communicate with the agent.
clearSpeak ( )	Removes all added properties.
getInstructions (*Identifier*)	Returns a list of instructions of a certain recipy.
getReq uirements (*Identifier*)	Returns a list of the requirements of a certain recipe.
getInstructionsCount (*Identifier*)	Returns the number of instructions of a certain recipe.
getTools ( )	Returns a list of all tools.
getToolFromAction (*Identifier*)	Returns a specific tool that can be used for a certain action.
getIngredients (*Identifier*)	Returns a list of ingrediënts of a certain recipe.
reset ( )	Resets the VRAEnvironment to it's initial state.

All actions as well as their arguments are case insensitive.

Percepts: The VRAEnvironment sends the following Percepts:

recipeselected (*Identifier* , *Identifier*)	Sent when a recipe was selected by the user. The first Identifier is the name of the recipe, the second is the title.
assistentselected (*Identifier*)	The user selected an assistent. The Identifier holds the name of an MSAgent character.
nocapacity (*Identifier*)	When the user tried to add an ingredient to a tool with insufficient capacity to hold this ingredient. The identifier holds the regarding tool.
toolselected (*Identifier* , *Identifier*)	When the user performed an action with a tool. The first Identifier is the action, the second is the tool.
ingredientadded (*Identifier* , *Numeral* , *Identifier*)	The user added an ingredient. The first parameter is the Ingredient, the second is the amount, the third is the Tool to which the ingredient was added.
ingredientremoved (*Identifier* , *Identifier*)	When the user removed an ingredient. The first parameter is the regarding ingredient, the second is the tool from which it was removed.
toolselected (*Identifier*)	Sent when the user selected a tool.
tooldeselected (*Identifier*)	Sent when the user deselected a tool; when it's set back into the closet.
ingredientselected (*Identifier*)	Sent when the user selected an ingredient.
speak (*Identifier*)	When the user "spoke" a sentence with the selection box.

Environment-management: There are no specific environment-management options.

BIJLAGE E

Utrecht, 30 mei 2010

Een beknopte inleiding tot

Java Native Interface

Hagenaars, Ramon

Inleiding

Een van de voornaamste voordelen van de moderne programmeertaal Java is de *platformonafhankelijkheid*. Ontwikkelaars hoeven, bij het schrijven van een applicatie, geen rekening te houden met platform specifieke aspecten, zoals de GUI, IO-onderdelen, etcetera. Hiertoe wordt broncode van Java-programma's gecompileerd naar Java bytecode, in plaats van platform specifieke machinecode. Java-applicaties draaien op een *Java Virtuele Machine* (*JVM*), die deze bytecodes uitleest, en hebben hierbij toegang tot de standaard bibliotheek van Java.

Het zou echter wenselijk kunnen zijn om technieken te gebruiken die de Java bibliotheek zelf niet bevat. Een programma dat opgesteld is in de instructietaal van de hardware, waarop ook de JVM draait, wordt ook wel een *native* programma genoemd.

De techniek, die gebruikt kan worden om een brug te slaan tussen de JVM en native programma's, is de zogenaamde *Java Native Interface*, afgekort als *JNI*. Op deze "brug" geldt een tweerichtingsverkeer; van de JVM naar de native programma's en andersom. Dit document zal zich echter alleen richten op het meest voor de hand liggende gebruik van JNI: het gebruikmaken van native bibliotheken vanuit de JVM.

Een native programma aanroepen

Om een Java-applicatie toegang te geven tot een native techniek, dienen de volgende stappen ondernomen te worden:

I. Definieer de native methoden;
II. Schrijf een native programma;
III. Laad dit programma in de JVM;
IV. Roep deze methoden aan.

We bespreken deze stappen hieronder in het kort.

I. Definieer de native methoden

Het gebruik van de JNI is te vergelijken met dat van de reguliere Java-*interface*. Bij een interface worden slechts de naam, argumenten en returntype van de methoden gedefinieerd. Bij de Java Native Interface, de naam verraadt het al, is dit hetzelfde geval. Van de externe bibliotheek hoeven slechts de methoden gedefinieerd te worden, zodat deze aangeroepen kunnen worden. De inhoud van deze methoden staat natuurlijk gespecificeerd in het native programma zelf.

Het definiëren van een native methode kan op de volgende manier:

```
private native void sayThis(String s);
```

Bedenk hierbij dat de gedefinieerde methode exact overeen moet komen met de methode in de externe bibliotheek. Wanneer de naam, het returntype of de argumenten van de in Java gedefinieerde native methode niet overeenkomt met een methode in een geladen externe bibliotheek, verschijnt tijdens het uitvoeren van het Java-programma, een *UnsatisfiedLinkError*. Wanneer deze *Exception* zich voordoet, dient dus na te worden gegaan of de native bibliotheek wel geladen wordt en of de methoden hierin juist zijn gedefinieerd in Java.

II. Schrijf een native programma

Het schrijven van een native programma kan in verschillende programmeertalen. Voor de meest populaire programmeertalen, C en C++, zijn speciale tools ontwikkeld om dit te vergemakkelijken, bijvoorbeeld de *Javah.* Echter kan elke programmeertaal gebruikt worden die het genereren van native programma's en het aanspreken van dynamische bibliotheken ondersteunt.

We zullen in dit voorbeeld gebruik maken van Javah en de programmeertaal C. Hiertoe dient eerst de geschreven Java-class gecompileerd te worden, waarna we Javah gebruiken om een header file te genereren:

```
javac MyClassWithNativeMethods.java

javah  MyClassWithNativeMethods
```

Het gecreëerde header-bestand zal de naam van de klasse dragen met een .h extensie. In ons geval wordt dit dus: *MyClassWithNativeMethods.h*. De inhoud van dit bestand zal er dan als volgt uit zien:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class Test */

#ifndef _Included_MyClassWithNativeMethods
#define _Included_MyClassWithNativeMethods
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     MyClassWithNativeMethods
 * Method:    sayThis
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_MyClassWithNativeMethods_sayThis
  (JNIEnv *, jobject, jstring);

#ifdef __cplusplus
}
#endif
#endif
```

In dit header-bestand wordt alle informatie gedefinieerd die we nodig zullen hebben bij het schrijven van ons native programma.

Hieronder een voorbeeld van een implementatie van de native methode:

```
#include <jni.h> // Nodig om de JNI te kunnen gebruiken
#include <stdio.h> // Nodig voor de C-specifieke printf-operatie
#include "MyClassWithNativeMethods.h" // Het eerder gecreëerde header-bestand

JNIEXPORT void JNICALL
Java_MyClassWithNativeMethods_sayThis(JNIEnv *env, jobject obj, jstring str)
{
  printf("%s\n", str);
  return;
}
```

Vervolgens moet bovenstaande C-code worden gecompileerd en dient een DLL gegenereerd te worden. Wanneer we bijvoorbeeld de *GCC* zouden gebruiken, kan dit op de volgende manier:

```
gcc -c  MyLibrary.c

gcc -shared -o MyLibrary.dll MyLibrary.o MyClassWithNativeMethods.h
```

Als uitvoer zouden we dan het bestand *MyLibrary.dll* moeten krijgen: onze native bibliotheek.

III. Laad dit programma in de JVM

Het laden van een native programma kan in Java op de volgende manier:

```
System.loadLibrary("MyLibrary.dll");
```

We zien hier dat onze bibliotheek ingeladen wordt. De methode *System.loadLibrary* verwacht het complete pad naar de bibliotheek die geladen moet worden. Het laden van een externe bibliotheek hoeft slechts éénmaal te gebeuren. Hiertoe kan bovenstaande code ingebed worden in het statische deel van een klasse:

```
static
{
  System.loadLibrary("MyLibrary.dll");
}
```

IV. Roep deze methoden aan

Het aanroepen van de methoden met het keyword *native* verschilt niet met het aanroepen van van enige reguliere Java methode. Wanneer we dus de in vorige paragraaf gedefinieerde methode willen aanroepen, kan dit als volgt:

```
sayThis("Hello?");
```

Bronnen

http://java.sun.com/docs/books/jni/html/start.html
http://www.cygwin.com/cygwin-ug-net/dll.html
http://nl.wikipedia.org/wiki/Javah
http://en.wikipedia.org/wiki/Java_Native_Interface