Afstudeerstage

Framework voor het genereren van een autonoom dialoogsysteem met een specifiek dialoog

Auteur:
Raymond Siudak

30-05-2011

Raymond Siudak

Voorwoord

Al snel was ik gefascineerd van het feit dat de wereld om ons heen vertaald kan worden in 'bits & bites'. De mogelijkheid bood zich aan om te gaan studeren en die heb ik genomen. In het begin van de opleiding raakte ik al snel geïnteresseerd in autonome systemen en hoe deze de wereld kunnen interpreteren en beïnvloeden. Op dit gebied heb ik mij kunnen specialiseren.

Het resultaat hiervan is de scriptie voor u, deze is geschreven ter afsluiting van de bachelor 'Technische Informatica' aan de Hogeschool Utrecht. De afstudeerstage heeft plaatsgevonden in de periode van februari 2011 tot en met juni 2011 bij TNO binnen het kenniscentra 'Behavioral and Sociatal Sciences' op het expertisegebied 'Perceptual and Cognitive systems'.

Mijn gehele opleiding en het schrijven van deze scriptie was een plezierige en vooral leerzame ervaring. Graag wil ik van deze gelegenheid gebruik maken om mijn dank uit te spreken aan de mensen die mij hebben begeleid en ondersteund.

Op de eerste plaats mijn lieve vriendin en partner Laaila, voor haar geduld en support, zonder haar was dit niet mogelijk geweest. Op de tweede plaats Maarten Aalbers, voor het altijd gaan voor een goed resultaat en de succesvolle projecten die we hebben afgerond. Daarnaast wil ik Leo Van Moergestel bedanken voor 'Keeping an open mind' en zijn enthousiaste begeleiding.

Als laatste wil ik graag mijn dank uitspreken voor de originele bedenkers en ontwikkelaars van Ashley. Tina Mioch, voor haar kritische begeleiding gebracht met een glimlach en veel enthousiasme en Willem Van Doesburg voor het mogelijk maken van deze stage. Ik heb er veel van geleerd!

Abstract

Deze scriptie beschrijft een oplossing, die het mogelijk maakt een intelligente software agent op een eenvoudige en snelle manier te programmeren. In de oude situatie koste het programmeren veel tijd en vereiste het een hoog technische kennisniveau.

De resultaten beschreven in dit document zijn van toepassing op het door T.N.O. ontwikkelde Ashley. Zij is een interface voor een complexe agent, met de agent kan gecommuniceerd worden op een intuïtieve manier. Zij is een persoonlijke assistent, ze 'kent' de gebruiker en helpt hen de technologie om hen heen te hanteren. Ashley is een gedistribueerd systeem, bestaande uit een AI component met dialoogsysteem, een representatie van een virtueel karakter, een spraakherkenning- en spraaksyntheseserver.

Om tot de oplossing te komen is er onderzoek gedaan naar dialoogsystemen, intelligente agents en het modeleren van dialogen. Gelijktijdig aan het onderzoek is de interne werking van het systeem blootgelegd. Eén resultaat van dit onderzoek is, dat de verschillende dialoogtypen geïmplementeerd kunnen worden volgens een dialoogmodel. Het dialoogmodel beschrijft een vijfstappenplan dat is vertaald naar een concept waarin alle gewenste functionaliteiten zijn meegenomen. Daarnaast is er een methode voor het modelleren van deze dialogen ontwikkeld. De dialoogdata kan ingevoerd worden in XML volgens de ontwikkelde dialoogspecificatie. In het XML bestand is het dialoog met alle componenten opgeslagen. Het bestand dient als input voor de ontwikkelde Dialoog generator, deze genereert het dialoog in de source van het dialoogsysteem, compileert het geheel en maakt er een demo bestand van. De dialoog moet eerst getest worden voordat het in gebruik genomen kan worden, hiervoor is een gebruikersinterface ontwikkeld die de belangrijke data en componenten op een overzichtelijke manier representeert.

Door deze resultaten is Ashley naar een hoger niveau gebracht. Het implementeren van dialogen is toegankelijk gemaakt voor dialoogontwikkelaars zonder bijzondere technische kennis. Daarnaast zijn er veel mogelijkheden ontstaan voor uitbreiding in het gedistribueerde systeem.

Contents

1 Inleiding

Bij wetenschappelijk onderzoek op het gebied van de computer wetenschappen, weegt het onderzoek op zich, wel eens zwaarder dan de implementatie. Door tijdgebrek en/of financële redenen, kan het voorkomen dat de implementatie snel afgerond of minimaal geïmplementeerd moet worden. Helaas zijn de middelen later ook niet altijd beschikbaar, omdat men al weer op andere projecten is ingezet.

Een dergelijk probleem doet zich ook voor bij T.N.O., waar de afdeling Human Factors een virtuele agent heeft ontwikkeld. De agent wordt Ashley genoemd en kan ingezet worden op gebieden als; de medische zorg, defensie of het onderwijs. Met Ashley kan je communiceren op een natuurlijk manier, hierdoor zou zij taken van mensen over kunnen nemen zoals; les geven, een diagnoses stellen of een procedure vertellen. Daarnaast kan Ashley ook hardwarecomponenten aansturen, bijvoorbeeld domotica, een UAV of een koffieapparaat.

Om Ashley gereed te maken voor een bepaalde context, moet zij beschikken over de juiste kennis van het onderwerp en beschikken over een dialoogstrategie. Het kost veel tijd om deze kennis in Ashley te stoppen en het vereist een hoog kennisniveau van de gebruikte technologieën. Hierdoor kan alleen een ingewijde ontwikkelaar Ashley gereed maken voor een nieuwe context. Het doel van dit onderzoek is het ontwikkelen van tools waarmee de kennis in Ashley gestopt kan worden, daarnaast wordt er gekeken naar hoe het testproces overzichtelijker gemaakt kan worden. Om deze doelstellingen te realiseren is er kennis nodig. Kennis van hoe een dialoog wordt opgebouwd, welke dialoogvormen moeten er verwerkt kunnen worden, hoe wordt de informatie op dit moment verwerkt, uit welke onderdelen bestaat het systeem, hoe werken deze onderdelen, etc.

Dit document beschrijft de weg naar een oplossing voor deze problemen, om te beginnen wordt er achtergrond informatie gegeven over T.N.O, de gebruikte technologieën en de probleemstelling. Hoofdstuk twee behandelt de aanpak, de inventarisatie van de wensen van de opdrachtgever en de gevormde visie die het project richting heeft gegeven. Hoofdstuk drie schetst de uitgangssituatie, uit welke componenten bestaat het huidige systeem en hoe worden deze gebruikt. Deze informatie is nodig om een goed beeld te schetsen. In hoofdstuk vier wordt een analyse gemaakt van de componenten die in een dialoog kunnen voorkomen. Deze analyse is input voor het concept dat in hoofdstuk vijf wordt toegelicht. De resultaten die hieruit leiden worden besproken in hoofdstuk zes. Afsluitend volgen de conclusies en aanbevelingen.

Opmerking: In dit document worden veel technische termen gebruikt, om de leesbaarheid te bevorderen is er een begrippenlijst toegevoegd. Deze is te vinden aan het eind van dit document. Alle begrippen die vermeld zijn in de begrippenlijst, zijn schuingedrukt. Wanneer dit document digitaal gelezen wordt, kan er op het begrip - wanneer deze voor het eerst voorkomt - geklikt worden om direct naar de beschrijving te springen.

1.1 Achtergronden

1.1.1 Over TNO

TNO[1] is een onafhankelijke onderzoeksorganisatie, die staat voor doelgericht innoveren. TNO past de opgebouwde kennis toe voor en met opdrachtgevers, hierbij kan gedacht worden aan: overheden, het MKB, grote bedrijven, dienstverleners en maatschappelijke organisaties. TNO is geen overheidsinstelling zoals vaak gedacht wordt, het is een zelfstandige instelling en eigen rechtspersoon volgens de TNO-wet.

Sinds januari 2011 hanteert TNO de organisatiestructuur van een matrixorganisatie. De meer dan 80 kennisgebieden zijn onderverdeeld in een drietal kennis centra: 'Technical Sciences', 'Behavioral and Societal Sciences' en 'Earth, Environment and Life Sciences'. Voor de buitenwereld voert TNO projecten uit die vallen onder één of meerdere van de zeven thema's. Te weten: gezond leven, industriële innovatie, integrale veiligheid, energie, mobiliteit, gebouwde omgeving en informatiemaatschappij.

Binnen het kennis centra 'Behavioral and Societal Sciences' bestaan er diverse expertise groepen. Eén daarvan is 'Perceptual and cognitive systems' waarbinnen de afstudeerder zijn plek heeft.

1.1.2 Ashley uw virtuele persoonlijke assistent

'Steeds minder mensen moeten steeds meer werk verzetten. Technologie kan een oplossing bieden, tenminste als werknemers er goed mee om gaan. Ook thuis en onderweg is technologie alomtegenwoordig. Hoe houden mensen de regie over alle technologie? Ashley kan helpen. Ashley is een virtuele persoonlijke assistent die mensen begeleidt in hun omgang met dagelijkse technologie'[10]

Figuur 1: Foto's van 2 Ashley opstellingen met visuele verschijning

Met Ashley kan je op een intuïtieve manier communiceren. Dit maakt een goede communicatie mogelijk, hierdoor kunnen de wensen, de mogelijkheden en de beperkingen van de mensen vertaald worden naar de technologie in hun omgeving.

Via Ashley zou je de verlichting in je huis kunnen aansturen. Ashley zou file informatie kunnen geven of de 'voor jou' interessante nieuws headlines kunnen voorlezen, tijdens het autorijden . Ashley zou je kunnen helpen met je huiswerk of je les kunnen geven.

Op verzoek van het ministerie van Economische zaken geeft TNO workshops waar filmpjes worden afgespeeld en demonstraties worden gegeven om de mogelijkheden van Ashley te demonstreren en te onderzoeken. Daarnaast wordt er op dit moment gewerkt aan andere functionaliteiten en toevoegingen, variërend van een Google Agenda koppeling tot emotieherkenning.

[1]Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek

1.1.3 Technologie

Ashley is een gedistribueerd systeem dat bestaat uit een visualisatie (weergegeven door een game engine), een spraakherkenning server en het AI *(Artificial Intelligence)* component. Het AI component bestaat voornamelijk uit een dialoogsysteem dat kan redeneren over de kennis van een gesprek, zoals: Wat is een vraag? Wat is mijn doel met het gesprek? Wat is het doel van mijn gesprekspartner? Wat is een antwoord? Naast kennis over het gesprek bevat de AI ook kennis over de wereld die de inhoud geeft aan een specifiek dialoog.

In de volgende sub-paragraven worden een aantal concepten beschreven, die in meer of mindere mate voorkomen in Ashley.

Agent
Een agent is een complexe entiteit die tot een zekere hoogte taken autonoom kan uitvoeren. Een object-geörienteerde taal laat zich beschrijven door operaties en attributen, een agent daarentegen wordt gedefinieerd door zijn gedrag[1].

BDI
Oftewel Belief-Desire-Intention, is een model - van grondlegger Micheal Bratman[2] - voor de praktisch redenatie van mensen. Rao and Georgeff hebben onder andere dit model in de praktijk gebracht voor het ontwikkelen van intelligente agents op basis van *BDI*[3].

Wat een agent aan kennis heeft over de wereld worden beliefs genoemd, de reden dat er is gekozen voor de benaming 'belief' (geloof) is dat de kennis niet altijd waar hoeft te zijn. Daarnaast heeft een agent desires (wensen) om bepaalde Goals uit te voeren. Een voorbeeld van een desire is 'vandaag gaan zwemmen' of 'vandaag gaan mountainbiken'. Een intention (intentie) is wat de agent heeft gekozen om te doen, hiervoor voert hij een plan uit. Een plan kan een serie stappen zijn om de intentie uit te voeren, bijvoorbeeld: de handelingen die nodig zijn om boven water te blijven en waar naar toe te zwemmen.

Dialoogsysteem
Dialoogsystemen worden op grote schaal ontwikkeld, voor zowel commerciële als voor educatieve en wetenschappelijke doeleinden. Een dialoogsysteem stelt de gebruiker in staat om te interacteren met een machine via natuurlijke taal, hiermee kun je bijvoorbeeld informatie op vragen of een diagnose laten stellen.

De toepassing bepaald grotendeels de mate van complexheid van input waar het systeem op moet kunnen reageren. Zo kan het zijn dat een systeem een serie vragen stelt waarbij alleen geluisterd wordt naar bepaalde woorden, bijvoorbeeld twee, Utrecht of trein. Dit wordt 'Geïsolleerde woord herkenning' genoemd.

Daartegenover staat dat het systeem antwoorden op open vragen kan verwerken, bijvoorbeeld op de vraag 'Hoe kan ik u van dienst zijn?'. Deze systemen gebruiken een strategie met een gemengd initiatief. Het systeem probeert door middel van het stellen van vragen, achter de intentie van de gebruiker te komen, deze aanpak brengt wel een hogere foutgevoeligheid met zich mee.

De mogelijkheden en problemen van dialoogsystemen worden nog op grote schaal onderzocht. Eén van de grootste uitdagingen is, hoe je een gesprek zo natuurlijk mogelijk kan maken. Een mens kan in een gesprek uitspraken maken die meerdere betekenissen kunnen hebben. De uitspraak kan dan alleen maar begrepen worden als deze wordt geplaatst in zijn context. Een ander onderzoeksgebied is: de verwachtingen van de gebruiker over het systeem te koppelen aan de mogelijkheden van het systeem. Zo kan iemand die bekend is met het systeem al weten wat de mogelijkheden zijn en iemand die onbekend is met het systeem geen idee hebben[4].

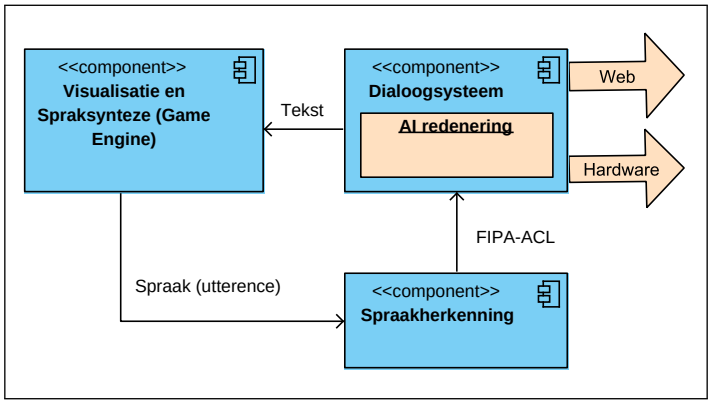
1.2 Projectopdracht

In dit hoofdstuk is de probleemstelling geformuleerd zoals deze is gegeven door de opdrachtgever, evenals de geëxtraheerde deelvragen en de doelstelling van het project.

1.2.1 Probleemstelling

Ashley bestaat uit meerdere componenten (Figuur 2), te weten een visualisatie (in een game engine), een spraakherkenning server en een dialoogsysteem. Het dialoogsysteem is het brein van Ashley, hierin zit de kennis over het gesprek met de gebruiker (wat is een vraag? Wat is een antwoord? Wat is mijn doel met het gesprek? Wat is het doel van mijn gesprekspartner?). Daarnaast zit er in het AI component ook kennis over de wereld, gecodeerd in een Prolog database. Omdat er zoveel kennis gerepresenteerd is in Ashley, is het relatief veel werk om Ashley klaar te stomen voor een nieuwe demo of toepassing.

Figuur 2: Visuele weergave van de componenten van Ashley



1.2.2 Vraagstelling

Uit de probleemstelling blijkt dat het programmeren van Ashley toegankelijker moet worden gemaakt en niet alleen voor techneuten. Daarnaast is het van belang de implemetatietijd te verminderen. Hierbij staat de volgende vraagstelling centraal:

Het vereist een hoog technische kennisniveau en veel tijd om Ashley gereed te maken voor een nieuwe toepassing, hoe kan het benodigde technische kennisniveau en de implementatietijd omlaag worden gebracht?

In beginsel is het van belang te onderzoeken wat een *BDI agent* precies is, wat een dialoogsysteem precies inhoud en hoe dialogen worden opgebouwd. Daarnaast is het van belang de componenten, de interne werking en de gewenste functionaliteiten van Ashley goed in beeld te hebben. Even belangrijk is het in beeld brengen van de gebruikers processen, hoe wordt een dialoog geïmplementeerd en hoe verloopt het testen. Pas dan kan er gewerkt worden aan een oplossing die het benodigde kennisniveau en de implementatietijd terugdringt.

1.2.3 Doelstelling

Om Ashley gereed te maken voor een nieuwe toepassing is een hoog kennisniveau van verschillende technologieën vereist, daarnaast is het testproces van een nieuwe dialoogtoepassing tijdrovend. Het doel van dit project is dit proces te vergemakkelijken en te versnellen, door het vereiste kennisniveau omlaag te brengen en het testproces te versnellen.

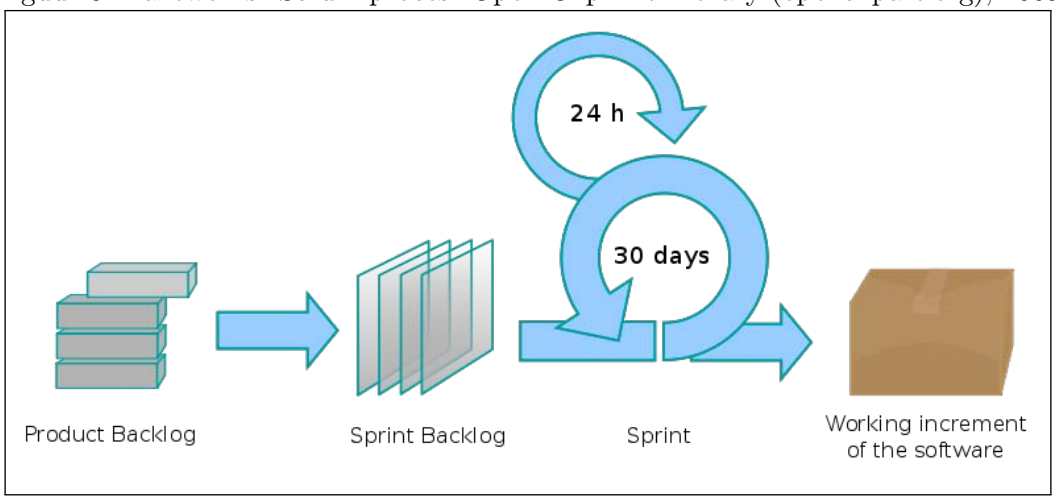
2 Aanpak

Om een goed inzicht te krijgen in de situatie, heeft er een uitgebreide periode van onderzoek plaatsgevonden. Daarin is de werking van het Ashley systeem uitgebreid gedemonstreerd en getest, de opdrachtgever en andere belanghebbende zijn geinterviewd en er is een visie gevormd die richting geeft aan het project. In dit hoofdstuk wordt eerst beschreven en beargumenteerd welke keuzes er zijn gemaakt met betrekking tot de gekozen projectmanagementmethode. Daarna worden er een aantal mogelijke oplossingen beschreven, met daarbij de gemaakte keuzes en afwegingen.

2.1 Scrum projectmanagement

Scrum is een raamwerk waarbinnen complexe producten ontwikkeld kunnen worden. Deze projectmanagementaanpak valt onder de categorie Agile -Methodes[6], dit houdt in dat er wordt gewerkt in iteraties en incrementen. Door de transparante ontwikkel aanpak, kan het proces continu gecontroleerd en bijgestuurd worden. De methode maakt een optimale flexibiliteit en productiviteit mogelijk.

Figuur 3: Lakeworks. Scrum proces. Open Clip Art Library (openclipart.org), 2009



Eén van de redenen waarom er is gekozen voor de Scrum methode, is dat een groot deel van de opdracht bestaat uit het doen van onderzoek, het eindresultaat krijgt pas in een later stadium meer vorm. De gestelde eisen zijn grotendeels niet-functionele van aard. Dit vraagt om een flexibele aanpak die het mogelijk maakt het proces continu bij te sturen. Daarom zijn lineaire projectmanagementmethoden, zoals de waterval methode uitgesloten.

Een andere afweging voor deze keus, is het zelf kunnen bepalen van de doorloop en het doel van iteraties, zolang het iteraties van twee tot vier weken zijn. Het doel en het resultaat van elke iteratie wordt bepaald in overleg met de opdrachtgever, hierdoor is er snelle en reguliere besluitvorming mogelijk. Andere *agile* methoden zoals DSDM (Dynamic Systems Development Method) hebben een meer gestructureerde fasering, hierdoor passen ze beter bij projecten met grotere project-teams of projecten met minder onbekenden en een concreter eindresultaat.

2.1.1 Scrum (Timeboxes)

In deze subparagraaf worden de zogenaamde Scrum 'Timeboxes'[6] in het kort toegelicht, dit geeft een beeld van de regelmaat in het project. Beginnend met de officiele kick-off van het project, ookwel 'Release planning bijeenkomst' genoemd. Het doel van de release planning is het opstellen van doelstellingen en een plan. Er wordt gekeken naar hoe de huidige visie omgezet kan worden in een succesvol product. Daarnaast worden de globale features en functionaliteiten vastgelegd, alsmede zaken als de belangrijkste risico's en de waarschijnlijke opleverdatum.

Wanneer het project officieel is begonnen, wordt er overgegaan naar het iteratieve proces. Een iteratie heet een 'Sprint'. Een Sprint is een tijdsframe variërend van twee tot vier weken. Om een nieuwe Sprint te starten wordt er eerst een 'Sprint Planningsbijeenkomst' georganiseerd. In deze bijeenkomst worden de doelstellingen van de Sprint vastgelegd. Er wordt gekeken naar de op te leveren producten, deze worden geprioriteerd of verder uit gedefinieerd. Aan het resultaat van de Sprint worden eisen gesteld, dit wordt de 'Definitie van Done' genoemd.

Tijdens de Sprints wordt er dagelijks een zogenaamde 'Daily Scrum Meeting' gehouden van ongeveer vijftien minuten. In deze meeting wordt er door elk teamlid antwoord gegeven op de volgende vragen: Wat heb je bereikt sinds de vorige meeting? Wat wil je bereiken tot de volgende meeting en zijn er mogelijk belemmeringen die dit in de weg staan?. Het project blijft hierdoor transparant, het bevorderd de communicatie en het geeft een goed overzicht voor iedereen.

Ter afsluiting van elke Sprint wordt er een 'Sprint Review' gehouden, aanwezigen hierbij zijn het project team, de 'Product Owner' en eventueel andere belanghebbende. Tijdens deze bijeenkomst wordt het gedane werk gedemonstreerd, er wordt geëvalueerd of de doelstellingen gehaald zijn en of het resultaat voldoet aan de gestelde eisen. Daarnaast worden eventuele problemen besproken en wat er is gedaan om deze te voorkomen of de impact te beperken. De feedback uit de review zal meegenomen worden naar een (mogelijk) volgende Sprint.

Voordat er over wordt gegaan naar de volgende Sprint, wordt er nog een zogenaamde *"Retrospective"* gehouden. Dit is een bijeenkomst waarin het team reflecteerd op het algemene proces, de gang van zaken en elkaar. De punten die hieruit komen worden meegenomen om het proces te verbeteren.

2.1.2 Kwaliteitsborging

De Scrum methode zorgt voor een transparant proces. De ingebouwde mechanismen; de Sprint Planning, de Daily Scrum, de Review en de Retrospective zorgen ervoor dat bijna dagelijks de risico's kunnen worden geanalyseerd en er tijdig kan worden ingegrepen. Voor elke Sprint zijn de doelen en de eisen vastgelegd die na elke Sprint worden geëvalueerd. Door het *agile* karakter van deze aanpak wordt de kwaliteit optimaal gewaarborgd [6].

2.2 Eisen inventarisatie & analyse

Een resultaat van de inventarisatie & analyse legt de basis voor het op te leveren geheel 'een gevormde visie voor het eindresultaat'. Een ander even belangrijk resultaat zijn de geanticipeerde risico's.

2.2.1 Eisen inventarisatie

Een veelgebruikte term in de *agile* projectmanagement methoden is *user story*[6]. *User stories* worden verkregen door een serie informele inhoudelijke gesprekken te voeren. Informatie uit deze gesprekken kan omgezet worden naar kleine (passend op een post-it) verhaaltjes van maximaal een paar zinnen. Deze zinnen vertellen wat de gebruiker wil bereiken. Uit deze zinnen kunnen vervolgens functionele en niet-functionele eisen en eventuele producten worden herleid. Onderstaand de overeengekomen lijst met user stories:

1. Ik als ontwikkelaar, zou graag een standaard *Prolog* kennisbase willen hebben voor kennis over de algemene dialoog.
2. Ik als ontwikkelaar, zou graag de mogelijkheid willen hebben een aparte *Prolog* kennisbase aan te maken voor specifieke kennis over het onderwerp.
3. Ik als ontwikkelaar, zou graag een goede scheiding willen hebben van de dialoog kennis en onderwerp kennis, in zowel Java als *Prolog*.
4. Ik als ontwikkelaar, zou graag inzicht willen hebben in wat dialoog kennis en onderwerp kennis is.
5. Ik als opdrachtgever, zou graag een zo generiek mogelijk dialoogsysteem willen hebben zonder het verlies van functionaliteit.
6. Ik als ontwikkelaar, zou graag willen dat er in het nieuwe systeem rekening is gehouden met het koppelen van externe modules/data.
7. Ik als tester, zou graag betere terugkoppeling willen krijgen over de onderliggende beslissingen van het systeem.
8. Ik als ontwikkelaar, wil graag beter kunnen zien wat de verschillende routes door de dialoog zijn.
9. Ik als tester, zou graag een stilte van meerdere seconden willen kunnen testen.
10. Ik als gebruiker, zou graag meer inzicht willen hebben in de werking van het dialoogsysteem en de demo-opstelling.
11. Ik als ontwikkelaar, zou graag in minder tijd een nieuwe dialoog willen implementeren.
12. Ik als tester, zou graag sneller meer van de dialoog kunnen testen.
13. Ik als gebruiker, zou graag een overkoepelend demobestand willen hebben, zonder losse files voor het eenvoudig starten van de demo.
14. Ik als gebruiker zou graag een editor willen hebben voor het koppelen van animaties aan uitingen.

2.2.2 Vertaling van wens naar visie

De geïnventariseerde *user stories* beschijven onderdelen van het probleem en wat de opdrachtgever graag terug zou willen zien in het systeem. Onder de *user stories* is een clustering aangebracht, de clustering is gedaan op basis van de gestelde doelen en de inhoud van de *user stories*. In de onderste twee rijen zijn de genummerde *user stories* te zien. In samenspraak met de opdrachtgever is de prioritering van de doelen bepaald en zo is de basis gelegd om een visie te kunnen vormen.

Tijdens het traject van visievorming hebben een aantal mogelijke oplossingen de revue gepasseerd. De afwegingen die hierin gemaakt zijn worden kort beschreven. De genoemde afweging tussen de oplossingen resulteerd in een visie voor het te ontwikkelen systeem.

Table 1: Clustering gebruikerswensen

1 Versnel en vergemakkelijk het aanmaken/ bewerken van een dialoog in het dialoog systeem			**2** Versnel en vergemakkelijk het implementeren en testen van een dialoog in het dialoog systeem.
1.1 Ontwikkel een raamwerk voor grammar editing en voor de mapping van FIPA ACL berichten naar Proposities. Prio: HOOG	**1.2** Ontwikkel een raamwerk voor *goal* editing, *Prolog* editing en voor de interactie hiertussen. Prio: HOOG	**1.3** Ontwikkel een raamwerk voor de bewerking van uitingen en animaties. Prio: LAAG	Prio: MIDDEL
3 — 4 — 5 — 6	1 — 2 — 3 — 4	14	7 — 8 — 9 — 10
11	5 — 6 — 11		11 — 12 — 13

Volledig generiek systeem
Een mogelijke oplossing is een dialoogsysteem wat eenmalig gecompileerd is. De dialoog kan opgebouwd worden in een grafische user interface. De output van de user interface is een XML bestand dat ingeladen kan worden bij het starten van het systeem. Met deze oplossing is een laag kennisniveau nodig vanwege het eenvoudig aanmaken en inladen van een dialoog. Een ander voordeel is de lage foutgevoeligheid, fouten kunnen namelijk niet voorkomen in de aangepaste code, de fouten zouden zich alleen voor kunnen doen in het ontwerp en foutieve invoer van de dialoog door de gebruiker.

Het nadeel van deze oplossing is dat er veel aanpassingen nodig zijn in het huidige systeem, een groot deel van het dialoogsysteem zou herschreven moeten worden. Daarnaast verminderd deze oplossing de flexibiliteit van het systeem, dit zou een beperkende uitwerking kunnen hebben op mogelijke toekomstige toepassingen en deze oplossing jammergenoeg snel overbodig maken.

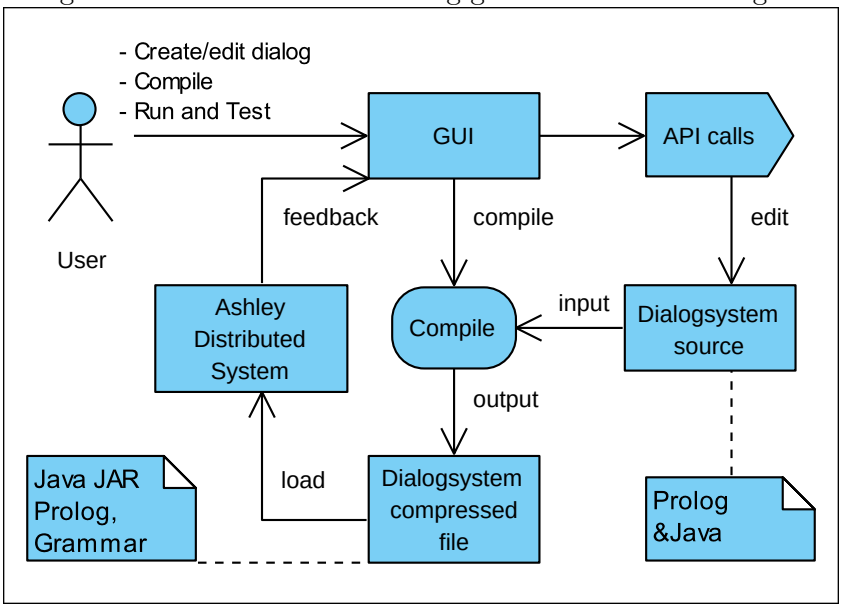
Semi generiek systeem
Het idee van deze opzet is in eerste instantie ook gebaseerd op een eenmalig gecompileerd dialoogsysteem. Echter bestaat het gecompileerde gedeelte alleen uit de kern 'het redeneerproces'. Om hiermee toch speciefiekere toepassingen mogelijk te maken, worden er templates gemaakt voor de onderdelen die specifieke dialoogkennis verwerken. Deze templates kunnen met behulp van goede documentatie, voor de specifieke toepassing worden ingevuld (geprogrammeerd en gecompileerd). De aangepaste templates kunnen dan in aan map geplaatst worden waar het dialoogsysteem deze kan vinden bij het opstarten. Het dialoogsysteem kan de templates dan weer inladen met behulp van een zogenaamde 'ClassLoader'. Hiermee wordt de flexibiliteit deels weer hersteld en is er duidelijke documentatie voor het aanmaken van een scenario. Een nadeel van deze oplossing is dat het kennisniveau niet echt omlaag wordt gebracht, er moet een vrij specifieke ontwikkelomgeving opgezet worden en de gebruiker moet een redelijke kennis hebben van Java en *Prolog*. Ook is de winst in snelheid minimaal en er is weinig toegevoegde waarde boven dat van een goede beschrijving voor het toevoegen van een scenario.

Dialoog generator
De genoemde bevindingen hebben geleid tot de dialoog generator. De dialoog generator is een framework bestaande uit een API (Application Programming Interface) en een gestructureerde manier voor het ontwerpen en invoeren van een dialoog. De API is voorzien van het gereedschap om de source van het dialoogsysteem te bewerken en te compileren. Dit concept kan beschreven worden met de volgende metafoor: De dialoog generator is een soort GUI Builder, alleen dan voor een dialoog en niet voor een GUI. Dit wil zeggen dat zonder bijzonder kennis over de interne werking, een dialoog aangemaakt kan worden. De generator neemt de originele source als input en genereert daarin de code voor de nieuwe dialoog.

2.2.3 Visie

De gevormde visie is visueel weergegeven in Figuur 4. De visie is een samenstelling van de beschreven 'dialoog generator' in 2.2.2 en de gestelde eisen die betrekking hebben op het testen en inladen van een dialoog. In de oude situatie werdt er getest door systeem informatie te printen naar een log bestand. Deze informatie is goed, alleen moeilijk te volgen omdat het continu afgedrukt blijft worden tijdens het runnen van het systeem. Dit is in de visie meegenomen als terugkoppeling naar een gebruikers interface tijdens het runnen van het systeem.

Figuur 4: Visie: GUI met dialoog generator en test weergave



2.2.4 Scope

De opdracht van deze stage is een duo opdracht, dit betekend dat twee personen deze opdracht uit voeren. De besluiten rond de genoemde oplossingen die worden beschreven in deze scriptie, zijn gemaakt in samenspraak met collega afstudeerder Maarten Aalbers. Rondom de kern van de opdracht is er een goede verdeling gemaakt van de componenten. Maarten heeft zich meer gericht op de spraakherkenning, de verwerking van de herkende spraak en de invoer van de gebruiker. Ik heb mij voornamelijk gericht op het redenatieproces in zowel Java als Prolog en de generatie van output.

2.2.5 Risico's

Een ander onderdeel van de analyse is het bepalen van risico's. De risico's kunnen voorkomen in meer of mindere mate. het dialoogsysteem is niet uitgebreid gedocumenteerd en het aanleren van nieuwe technieken kan meer tijd kosten dan verwacht. Hierdoor bestaat het risico op inschattingsproblemen omtrent tijd en de hoeveelheid werk die nodig is. Om de kans op opreden van deze risico's zo klein mogelijk te houden, is een goede samenwerking en beschikbaarheid van de betrokkenen vereist. Wanneer een risico optreed kan er direct hulp gezocht worden bij de originele ontwikkelaars.

2.3 Activiteiten en milestones

Om het resultaat op tijd en met goede kwaliteit op te leveren, hebben er een aantal kernactiviteiten en belangrijke momenten plaats gevonden. Dit hoofdstuk beschrijft in het kort de project activiteiten, gevolgt door een overzicht van de belangrijke beslismomenten.

2.3.1 Activiteiten

De activiteiten laten zich onderverdelen in drie categorieën: projectmanagementactiviteiten, onderzoeksactiviteiten en technische activiteiten. De rol van deze activiteitn wordt hieronder kort toegelicht.

De timeboxes beschreven in paragraaf 2.1.1 bepalen de regelmaat van het uit te voeren project op gebied van projectmanagement.

Onderzoek is een groot onderdeel van dit project. In beginsel is de gewenste uitkomst nog van relatief vage aard. Er is eerst onderzocht hoe het systeem werkt en in elkaar steekt. Daarna is er onderzocht welke verbeteringen er gemaakt kunnen worden met betrekking tot de mate van generiekheid. Er is gekeken naar wat er al beschikbaar is op dit gebied en er zijn nieuwe technologieën aangeleerd.

Het dialoogsysteem is een autonoom systeem gebaseerd op Java. Afhankelijk van het geïmplementeerde scenario zal het systeem met behulp van redenatie in *Prolog*, zelf bepalen hoe het dialoog verloopt. Voor de spraakinterpretatie wordt het *SRGS* formalisme gebruikt. De *ASR* server verzorgt deze interpretatie en zet deze om in commando's die zinvol zijn voor het dialoogsysteem. De commando's worden omgezet naar zogenaamde *FIPA* berichten en worden verstuurd als input naar het dialoogsysteem. Voor het genereren van een nieuw scenario voor Ashley wordt er programmatuur bewerkt en geschreven volgens de genoemde standaarden. Voor het documenteren van processen en de interne logica van systeem componenten, is de UML (Unified Modeling Language) notatie gebruikt.

2.3.2 Mijlpalen en belangrijke data

Om het succes van dit project te garanderen zijn er verschillende beslismomenten opgenomen, de zogenaamde 'milestones'. De milestones bieden ieder een go/no go-moment, dit is een middel om de koers van het project tijdig bij te stellen. Tabel 2 geeft een overzicht van deze milestones.

Table 2: Mijlpalen en belangrijke data

Datum	***Milestone***	***Omschrijving***
17 maart 2011	Inleveren Plan van Aanpak	
25 maart 2011	Review Sprint 1	Onderzoek & inventarisatie
11 april 2011	Review Sprint 2	Mapping dialoogproces
6 mei 2011	Review Sprint 3	Bouwstenen definieren
30 mei 2011	Review Sprint 4	Concept ontwikkeling
30 juni 2011	Eind Sprint 5	Implementatie

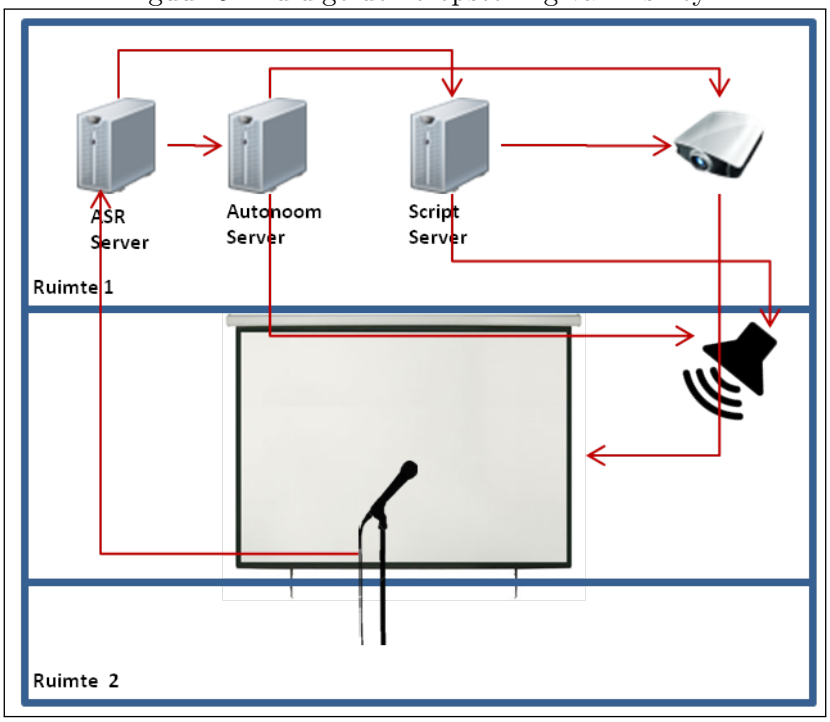
3 Uitgangssituatie

In paragraaf 2.3.1 is aangegeven dat onderzoek een groot deel is van dit project. Een onderdeel van het onderzoek is het in kaart brengen van Ashley en dan voornamelijk het dialoogsysteem. Dit hoofdstuk beschrijft de opstelling en algemene werking van het systeem. Gevolgd door een uitgebreide beschrijving van de architectuur en andere relevante onderdelen.

3.1 De opstelling van het systeem

In figuur 5 is de opstelling te zien. De opstelling wordt gebruikt over twee ruimtes: één ruimte voor de interactie en één 'Behind the scenes' ruimte waar de technische opstelling staat. De ruimtes worden gescheiden door een muur waarin een scherm verwerkt is, het scherm wordt van achter geprojecteerd zodat de verlichting in de interactie ruimte geen invloed heeft op het beeld. Ook zijn er drie server te zien in de weergave: één voor de spraakherkenning en twee servers die beide een demo kunnen draaien. De ene demo verloopt volgens een script en de ander is autonoom. Beide versies hebben ieder hun eigen opstelling en software componenten.

Figuur 5: Huidige demo opstelling van Ashley



3.1.1 Dialoog scenario volgens script

Het woord script zegt natuurlijk al genoeg. Het is een voorgeprogrammeerd dialoog, dat wil zeggen dat er van te voren is aangegeven waar Ashley naar luistert, waar zij op reageert en wat haar uitdrukkingen zijn. De voornaamste onderdelen van deze versie zijn: de spraakherkenning, de spraaksynthese en het virtuele karakter. De game engine zorgt voor het aflopen van het script, het geeft de spraakherkenningserver opdracht om naar bepaalde woorden te luisteren. De game engine is ook verantwoordelijk voor de spraaksynthese en de visuele weergave van Ashley.

3.1.2 Autonoom dialoog scenario

Het belangrijkste verschil tussen de scripted versie en de autonome versie is, dat hier geen script wordt gebruikt. Er is geen volgorde gedefinieerd voor de dialoog, het systeem moet op alle (gedefinieerde) uitspraken van de gesprekspartner kunnen reageren, onafhankelijk van de fase van de dialoog. Het script is dus vervangen door een dialoogsysteem. Dit dialoogsysteem redeneert over informatiebehoeftes op verzoeken van de gebruiker en de fase van de dialoog. Op basis van de verzoeken van de gebruiker en zijn eigen informatie behoeftes, zal het systeem output genereren in de vorm van vragen, antwoorden en animaties.

3.2 Dialoogsysteem ontwerp

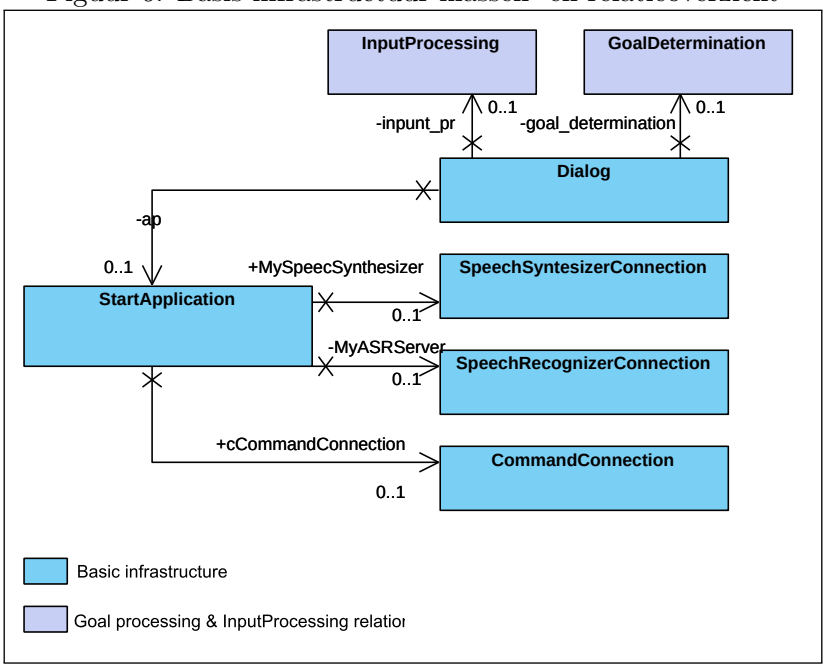
In deze paragraaf wordt de architectuur van het dialoogsysteem beschreven. Van het originele klassen diagram is een abstractie gemaakt dat een globaal beeld geeft van de componenten. In bijlage A zijn twee *sequence diagrammen* te vinden, die een globaal beeld geven van de het verwerkingsproces van het dialoogsysteem.

Het dialoogsysteem bestaat uit een aantal globale onderdelen, te weten: basis infrastructuur, input processsing, *goal* processing en de logische redenering. De onderdelen worden in het kort beschreven om inzicht te geven in de opbouw en de werking van het dialoogsysteem.

3.2.1 De basis infrastructuur

De basis infrastructuur is verantwoordelijk voor het opstarten, het afsluiten en het beheren van de netwerkverbindingen met de spraakherkenning- en de spraaksynthesemodule. Figuur 6 geeft een compact overzicht van de klassen behorende tot de basis infrastructuur. De klasse StartApplicatione is de main klasse en deze zorgt voor het instantieren van het geheel. De Dialog klasse is het hart van het dialoogsysteem, deze verzorgt de cyclus van het dialoogsysteem. De SpeechRecognizer- en de SpeechSyntesizerConnection klassen verzorgen de connecties en het protocol met de spraakherkenning- en de spraaksynthesemodule. De CommandConnection klasse luisterd op het netwerk voor een 'shutdown' commando. Er wordt een shutdown commando gegeven wanneer Ashely uitgeschakeld wordt.

Figuur 6: Basis infrastructuur klassen- en relatieoverzicht



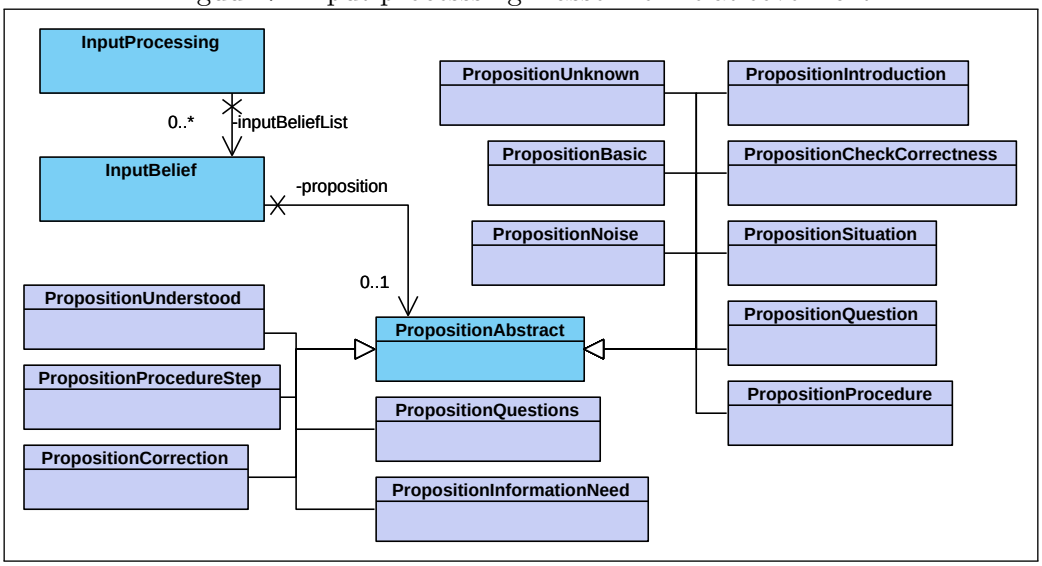
3.2.2 Input processing

In figuur 7 is een overzicht te zien van de klassen die belangrijk zijn voor het verwerken van de input. Wanneer de spraakherkenner een uitspraak heeft herkend zal hij een *FIPA* bericht sturen. De input wordt een *utterence* genoemd en wordt opgeslagen als string in een wachtrij. De InputProcessing klasse zet deze string weer om in een inputbelief. Tabel 10 laat zien wat de opbouw van een *inputbelief* is met daarbij een voorbeeld. Het voorbeeld beschrijft een inputbelief dat toegevoegd wordt als de gesprekspartner 'Hallo' tegen Ashley zegt.

Table 3: De opbouw van een Inform bericht met voorbeeld

Opbouw	**Voorbeeld**
<speechacttype> </xml content> </speechacttype>	<inform> <introduction> greeting </introduction> </inform>

Figuur 7: Input processsing klassen- en relatieoverzicht



Vervolgens wordt de *inputbelief* geclassificeerd naar speechact, de gebruikte *speechacts* zijn te zien in het model weergegeven in figuur 8. Aan de hand van het type *speechact* zal de XML parser de content parsen en de juiste *propositie* kiezen. De gekozen *propositie* vormt de basis van het redeneerproces, wanneer het systeem bijvoorbeeld een vraag heeft gesteld aan de gebruiker en er komt een antwoord binnen, dan wordt er een *goal* geselecteerd die deze informatie verwerkt. De *propositie* bevat de informatie die nodig is de *goal* uit te voeren. Tabel 4 geeft een overzicht van de *proposities* waar mee gewerkt wordt in het dialoogsysteem.

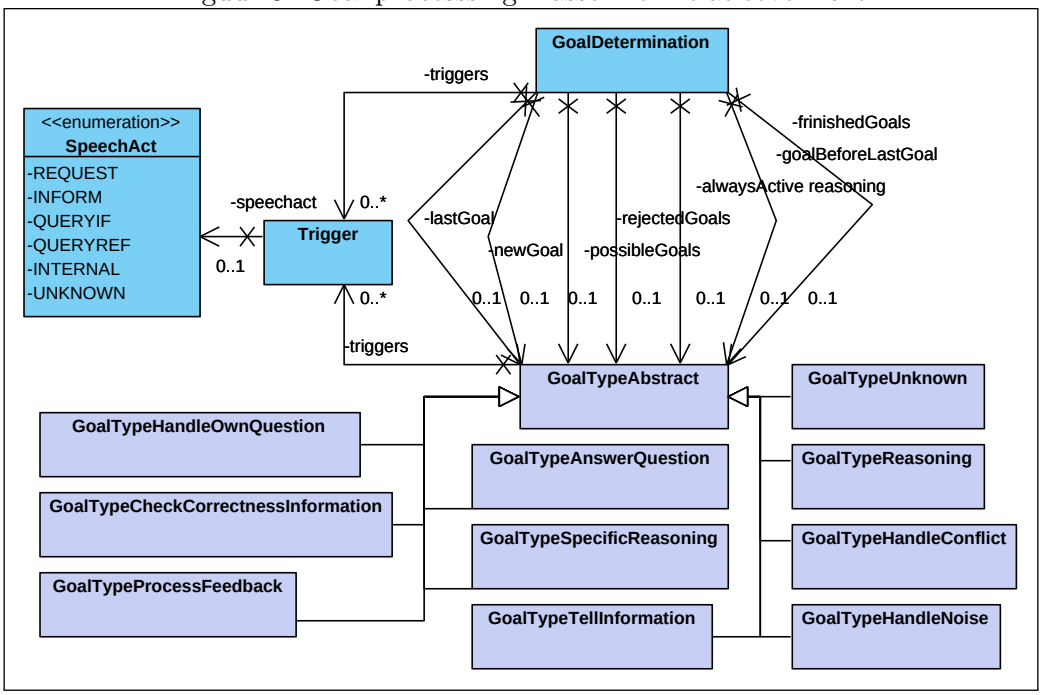
Table 4: Overzicht van de propositities met korte beschrijving

Klasse	**Omschrijving**
PropositionAbstract	Een template voor een *propositie*
PropositionBasic	Een *propositie* voor standaard conversatie
PropositionCheckCorrectness	Of een situatie of procedure correct is bevonden
PropositionIntroduction	Een introductie van de gebruiker
PropositionUnknown	Utterence niet bekend
PropositionSituation	Een beschrijving van de situatie
PropositionQuestion	Een vraag
PropositionNoise	Omgevings geluid of niet herkende spraak
PropositionInformationNeed	Een informatie verzoek
PropositionProcedure	Het onderwerp van conversatie, een procedure
PropositionQuestions	Of er nog vragen zijn over de informatie
PropositionUnderstoord	Of de informatie begrepen is

3.2.3 Goal processing

In figuug 8 zijn de onderdelen te zien die van belang zijn bij het selecteren van *goals*. De GoalDetermination klasse is hier grotendeels verantwoordelijk voor. De klasse heeft een lijst met *triggers*. De Trigger is bepalend voor het *goal* dat gekozen wordt. Dit gebeurd op basis van de combinatie *speechact* en *propositie* (behorende tot de *inputbelief*). Zie tabel 5 voor een overzicht van alle *goaltypes* met daarbij de functionele beschrijvingen.

Figuur 8: Goal processsing klassen- en relatieoverzicht



Wanneer de gewenste *goal* is bepaald wordt deze toegevoegd aan de actieve *goal* lijst. Wanneer er meerdere *goals* tegelijk actief zijn, wordt er aan de hand van een prioritering gekozen welke er eerst uitgevoerd wordt.

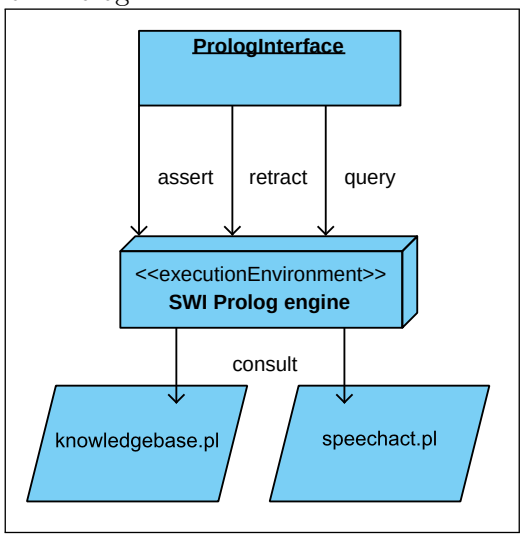
Table 5: Overzicht van de *goaltypes* met functionele beschrijving

Klasse	**Beschrijving**
GoalTypeCheckCorrectness-Information	Controleert of de gegeven informatie correct is. Er wordt een verificatie vraag gegenereerd en de *goal* blijft actief tot deze is beantwoord.
GoalTypeAnswerQuestion	De gestelde vraag wordt bepaald door de informatie in de *propositie*. Het antwoord wordt bepaald door de logische redenering.
GoalTypeHandleNoise	Deze *goal* reageert op ruis, eerst wordt er output gegenereerd die aangeeft dat er ruis heeft plaatsgevonden, na een bepaalde tijd zal er een verzoek voor input worden gegenereerd aan de gebruiker.
GoalTypeHandlerOwn-Question	Wanneer het dialoogsysteem beredeneerd dat er informatie nodig is om het gevraagde te kunnen afhandelen. Dan zal deze *goal* een vraag genereren.
GoalTypeProcessFeedback	Deze *goal* verwerkt informatie die nodig is voor de kennis opbouw, bijvoorbeeld een antwoord op een vraag.
GoalTypeReasoning	Deze *goal* is verantwoordelijk voor het redeneerprocess en draagt zorg voor het bijwerken van de opgebouwde kennis.
GoalTypeSpecificReasoning	Op het moment dat er een verzoek wordt gedaan, zal deze *goal* kijken of er informatie nodig is om het verzoek uit te kunnen voeren. Wanneer er informatie nodig is zal deze *goal* een *inputbelief* aanmaken die het verkrijgen van informatie afhandelt.
GoalTypeTellInformation	Deze *goal* zorgt voor het vertellen van de gevraagde informatie,de gevraagde informatie kan bestaan uit één zin of uit bijvoorbeeld een procedure van meerdere stappen. Deze *goal* zal alle stappen doorlopen en de output genereren.
GoalTypeUnknown	Lege *goal* gebruikt om een goal te instantiëren.

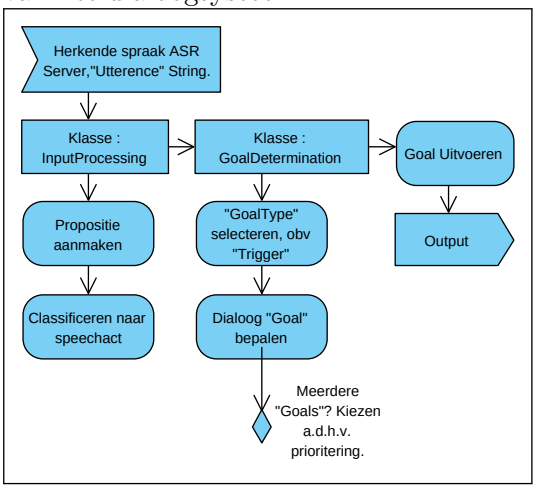
3.2.4 Logische redenering

De uitvoer en uikomst van een *goal* is niet altijd hetzelfde, dit wordt bepaald door de opgebouwde kennis over het gesprek. De redenering over de opgebouwde kennis wordt gedaan in *prolog*. De klasse PrologInterface gebruikt een bidirectionele interface om te interacteren tussen Java en Prolog. De interface is een Java bibliotheek genaamd JPL(a Java Interface to Prolog) en is een toevoeging aan de Prolog engine *SWI-prolog*[9]. De interface maakt het mogelijk om feiten te vervangen, de waarheid van feiten te toetsen of feiten op te vragen uit een *Prolog* database. Figuur 9 geeft een weergave van de componenten die nodig zijn om te interacteren met *Prolog*. Te zien is de PrologInterface klasse, de SWI-Prolog engine en de twee databases. De database knowledgebase.pl bevat alle kennis over de wereld en de regels om te redeneren over de dialoog. De database speechacts.pl bevat alle output zinnen die het systeem kan produceren.

Figuur 9: Weergave interactie tussen Java en Prolog



Figuur 10: Het interne afhandelingproces van het dialoogsysteem



In figuur 10 wordt een overzicht gegeven van het interne verwerkingsproces van het dialoogsysteem. Te zien is een standaard systeem-run. Er komt herkende spraak binnen in de vorm van een *utterence* vanaf de spraakherkenner. De InputProcessing klasse is verantwoordelijk voor het decoderen van de spraak. De GoalDetermination klasse selecteerd de uit te voeren *goals*. Afhankelijk van welke *goal* er is gekozen, zal deze output genereren.
Zoals te zien in de afbeelding wordt er een *propositie* aangemaakt, hiermee kan een Goaltype worden gekozen waarmee weer een *goal* wordt geselecteerd om uit te voeren.

4 Component Analyse

In het vorige hoofdstuk is de opzet van Ashley en het ontwerp van het dialoogsysteem uiteengezet. In dit hoofdstuk worden de elementen die Ashley moet kunnen verwerken in beeld gebracht. Er wordt gekeken naar hoe een dialoog geïnitieerd kan worden. Er worden diverse dialoog scenarios beschreven. Van de onderlinge overeenkomsten tussen deze scenarios wordt een abstractie gemaakt. Daarna wordt er een component uit het oorspronkelijke dialoogsysteem toegelicht dat zich uitstekend leent voor de afhandeling van de gemaakte abstractie.

Met behulp van enig literatuuronderzoek en de input van de analyse is er een *BDI*- en dialoogmodel opgesteld. Met deze modellen kunnen de zogenaamde bouwstenen voor het opbouwen van een scenario worden bepaald.

4.1 Initiatie van de dialoog

Dialoogsystemen bestaan er in verschillende vormen, zo heb je *IVR* en *QA* systemen. Dit zijn typische mens-machine dialoogsystemen, de interactie van deze systemen wordt beperkt door het doel en het ontwerp van het systeem. Ashley beschikt over een mens-machine dialoogsysteem. Je kan niet zomaar over een onderwerp beginnen met Ashley als zij daar geen kennis van heeft. De kennis over de dialoog moet immers in Ashley gestopt worden. Gelukkig is Ashley meer dan alleen een dialoogsysteem, het dialoogsysteem is enkel onderdeel van het AI component. Ashley heeft bijvoorbeeld ook een gezicht, die non-verbale uitdrukkingen kan overbrengen. In de toekomst misschien ook ogen waarmee ze veranderingen in de omgeving kan opmerken, of gezichten en emoties kan herkennen. Ashley kan dus op basis van andere input -van bijvoorbeeld een sensor- opmerken dat iemand een ruimte binnen komt en zelf een dialoog initiëren.

Voor de initiatie wordt dan een *FIPA* bericht verstuurd vanaf een extern component naar het AI component van Ashley. *FIPA ACL* is voor het AI component vergelijkbaar met een hersensignaal. De signalen fungeren als redenatie basis voor zowel input als output. De signalen worden geclacificeerd als *speechact*(Zie figuur:8.

4.2 Scenario analyse

Om de onderlinge overeenkomsten tussen de scenario's te overzien worden er een aantal scenario's in kort beschreven. Elke beschrijving wordt ondersteund door een tabel en een visuele weergave. De tabellen laten een mogelijk conversatie verloop zien van het scenario. De afbeelding illustreert de conversatie met alle mogelijke paden en keuzes.

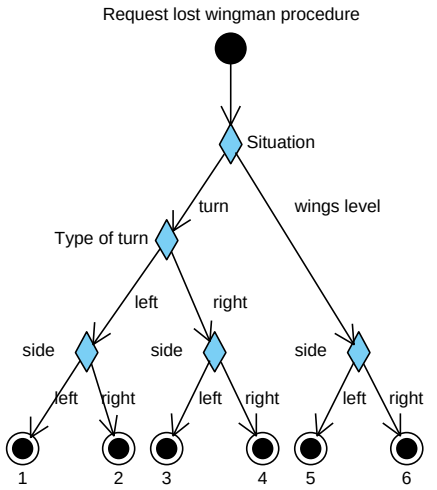
Op dit moment wordt het Ashley systeem voornamelijk gebruikt voor een scenario genaamd 'Lost Wingman'[11]. Om het gros van de mogelijke dialoog scenario's af te dekken zijn er twee scenario's bedacht en is er één nieuw scenario geïmplementeerd. Het voorbeeld van de 'Lost Wingman' schets eenmaal het scenario met tabel 6, de tabellen voor de andere scenario's zijn te vinden in bijlage B.

Opmerking: De eerste kolom van de scenario tabellen wordt ingevuld met een nummering. De betekenis van deze nummering wordt in de laatste paragrafen duidelijk gemaakt.

4.2.1 Lost Wingman

Dit is een scenario waarbij Ashley geïnstrueerd is over een zogenaamd 'Lost Wingman' procedure. De gesprekspartner kan Ashley vragen of ze de procedure begrepen heeft en vragen of ze de procedure kan vertellen. Om een procedure te vertellen moet Ashley natuurlijk weten over welke procedure het gaat. Daarnaast is er ook nog informatie nodig om de situatie te specificeren. Hiervoor zal Ashley een aantal vragen stellen, als alle antwoorden verwerkt zijn zal Ashley de situatie bevestigen en overgaan tot het vertellen van de procedure. Als laatste vraagt Ashley om een bevestiging.(Tabel 6)

Table 6: Lost Wingman conversatie analyse met een mogelijk verloop.

	Conversation	**Activity diagram**
1	1 U: Tell me the procedure pls?	
2	2 A: Turn or wings level? 3 U: A turn/wings level 4 A: What kind of turn/what side? 5 U: Right/left turn/side 6 A: Flying on what side? 7 U: Left/Right side	
3	8 A: I am in an inside/outside turn? or a am i flying left/right side? is this correct? 9 U: Correct	
4	10 A: I immediately safely sep.. 11 A: I maintain the safe... 12 A: You arrange a separate.. 13 A: I transition smoothly.. 14 A: I reduce power to...etc.. 15 A: And that's it.	
5	16 A: Is this correct? 17 U: Yes it is.	

4.2.2 Apparatenhulp

De apparatenhulp is een van de mogelijke toepassingen voor Ashley. In dit specifieke voorbeeld bestaat er alleen 'apparatenhulp' voor een koffiezetapparaat.

De gesprekspartner kan Ashley vragen te helpen met een apparaat. Voordat Ashley kan helpen moet ze weten over welk apparaat het gaat en wat voor een handeling er mee gedaan moet worden. Weer zal Ashley de nodige vragen stellen en de antwoorden hierop verwerken. Wanneer de informatie compleet is zal Ashley de bijbehoren stappen voor bijvoorbeeld koffiezetten vertellen. Als laatste vraagt Ashley om een bevestiging.(Zie bijlage B tabel 18)

4.2.3 Diagnose

Dit is een vereenvoudigd scenario van een hoofdpijn diagnose, later in het document wordt er een uitgebreidere versie gebruikt. In dit scenario worden er een aantal gesloten vragen gesteld door Ashley, waarop een positief of negatief antwoord gegeven kan worden. Wanneer alle informatie verzameld is, zal Ashley de diagnose en de behandeling vertellen. Ashley sluit de conversatie door te vragen of er nog wensen zijn. (Zie bijlage B tabel 19)

4.2.4 Home automation

Dit scenario is geschetst om rekening te houden met de koppeling van externe modules. In dit geval een home automation module voor het aanzetten van de verlichting in het pand. De gebruiker kan Ashley vragen het licht aan of uit te zetten. Ashley zal dan vragen stellen zoals: in welke ruimte, hoe fel en of het licht in andere ruimtes uitgezet moet worden. Wanneer de informatie compleet is zal Ashley de gevraagde actie uitvoeren. Dit kan bijvoorbeeld een bericht zijn naar een externe module die de home automation aanstuurt. In tegenstelling tot het eerder genoemde 'Apparatenhulp' scenario, voert dit scenario een actie uit in plaats van informatie vertellen. Ashley sluit de conversatie af door te vragen of er nog wensen zijn.(Zie bijlage B tabel 20)

4.2.5 Abstractie

Als we kijken naar de scenario's, dan is meteen duidelijk dat het conversatie doel in de scenario's veelal draait om het verkrijgen van informatie. Het zijn namelijk allen zogenaamde *Information seeking dialogs*[5].

Een conversatie heeft altijd een doel, oftewel een intentie. In alle scenario's valt op dat de informatie bepalend is voor de uitvoer van de intentie. Een intentie is bijvoorbeeld het stap voor stap vertellen van een procedure, of het licht in de ene kamer aanzetten en in de andere kamer uit.

Er is één component uit deze voorbeelden dat alleen terug te vinden is in het Lost Wingman scenario. De andere dialogen hebben deze optie simpelweg niet geïmplementeerd. Dit component is te zien in het voorbeeld (regel 8 t/m 10 tabel 6). Dit onderdeel bevestigd eigenlijk de situatie die is ontstaan na de informatie opbouw. Bij een 'correct' gaat het systeem over tot het uitvoeren van de intentie en bij een 'incorrect' gebeurt er niets.

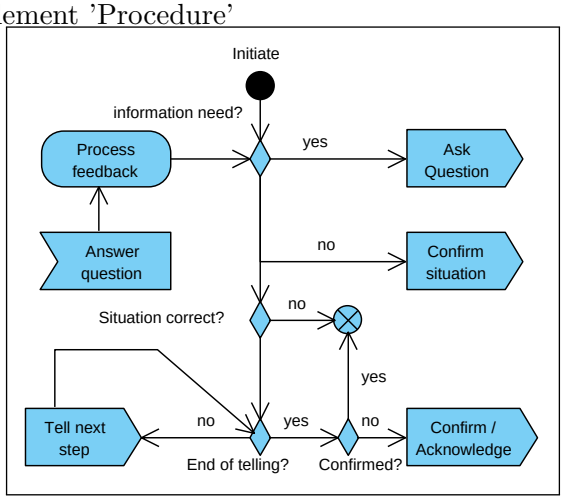
Op de laatste plaats is te zien dat alle scenario's afgesloten worden met een bevestiging of een vraag of er nog iets nodig is. Hierop kan een positief of negatief antwoord worden gegeven. Voor zowel het positieve als het negatieve antwoord kan een actie of output zin worden gedefiniëerd.

4.3 Het element 'Procedure'

Binnen het dialoogsysteem bestaat er een element 'Procedure', dit element is erg specifiek in zijn huidige vorm. Zoals te zien is in het klassendiagram in bijlage C, kom je dit element nagenoeg in het gehele systeem tegen.

Het element Procedure bestaat uit een aantal onderdelen; de initiatie, de informatie behoeften waar vragen voor worden gesteld, het vervullen van de behoeften door de antwoorden te verwerken, het bevestigen van de situatie, alle stappen van de procedure vertellen en als laatste de vertelde Procedure bevestigen. Figuur 11 schetst het interne verwerkingsproces van het 'Procedure' element. Het eerder genoemde Apparatenhulp scenario 4.2.2 is ook geïmplementeerd gebruikmakend van de onderdelen die de procedure verwerken. Plaatsen we de abstractie 4.2.2 in de context van dit element, dan is te zien dat deze goed op elkaar aansluiten. Met een paar aanpassingen is dit element dus in te zetten voor alle scenario's.

Figuur 11: Proces van het element 'Procedure'

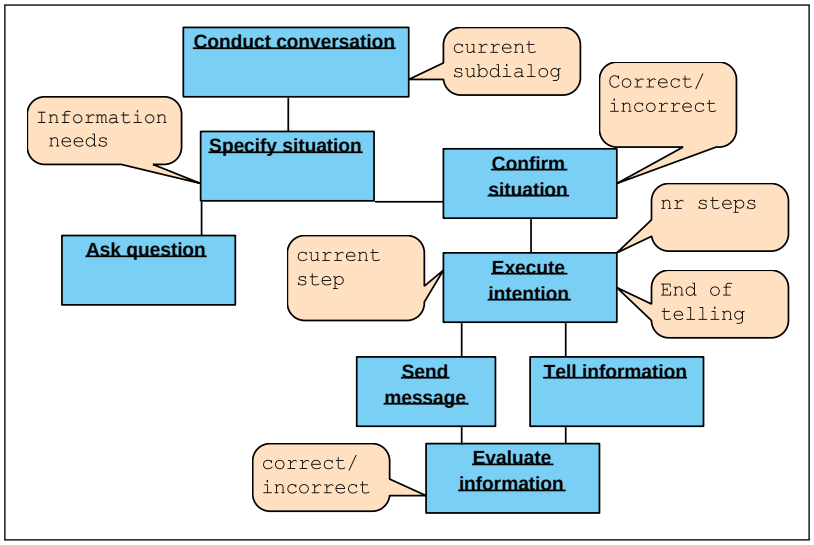


4.4 BDI model

Volgens de scenario analyse, kan de opzet van het dialoogsysteem weergegeven worden volgens het *BDI*[3] model in figuur 12. Aan de originele opzet is een aanpassing gemaakt. Exucte intention kan naast het vertellen van informatie ook berichten versturen die interne of externe acties teweeg kunnen brengen.

In de weergave zijn de 'beliefs' weergegeven met ballonnetjes, de 'desire' bestaat om de situatie duidelijk te krijgen, de 'intentie' is om de gevraagde informatie te vertellen of de nodige acties uit te voeren en de 'desire' bestaat om de vertelde informatie te verifiëren.

Figuur 12: Beliefs Desire Intention model van de scenario's



4.5 Dialoog model

Volgens de bevindingen van Giovanni Flammia, is een mens-tot-mens conversatie een gemeenschappelijke activiteit die ontleedbaar is in op zichzelf staande segmenten[5]. De grenzen van deze segmenten kunnen bepaald worden op verschillende manieren. Zo kan er bijvoorbeeld gezocht worden naar logische verbanden (Unconstrained segmentation), er kan van te voren een structuur voor de conversatie bepaald worden (Directed Linear Segmentation) of een hiërargische onderverdeling van taken en sub-taken in een conversatie (Directed Embedded Segmentation).

Dit onderzoek is zeer toepasbaar op de conversatie structuren van Ashley. De beschreven scenario's zijn goed in te delen volgens de hierargische onderverdeling van taken en sub-taken.

Table 7: Ashley dialoog model

Nr	Label
1	*Trigger*
2	Specify situation
3	Confirm situation (optional)
4	Execute intention
5	Evaluate intention

Zoals beschreven in de component analyse 4.2.5 zit er nauwelijks verschil in de basis van de genoemde scenario's. De verschillen zijn een bevestiging op de situatie, een evaluatie op de intentie en de uitvoer van de intentie. Deze onderverdeling is al aangebracht in de omschreven scenario's, de genummerde segmenten komen overeen met het dialoogmodel beschreven in het tabel links. In het dialoogmodel is te zien dat een conversatie altijd gestart wordt met een *trigger*, de herkende uitspraak van de gesprekspartner kan de trigger zijn of een externe bron zoals bijvoorbeeld een sensor. Daarna wordt de situatie achterhaald door middel van een vraag-antwoord serie, de situatie bepaald de uit te voeren intentie. Een intentie kan zijn, een procedure stap-voor-stap vertellen of bepaalde actie uitvoeren. Als laatste kan de intentie geëvalueerd worden.

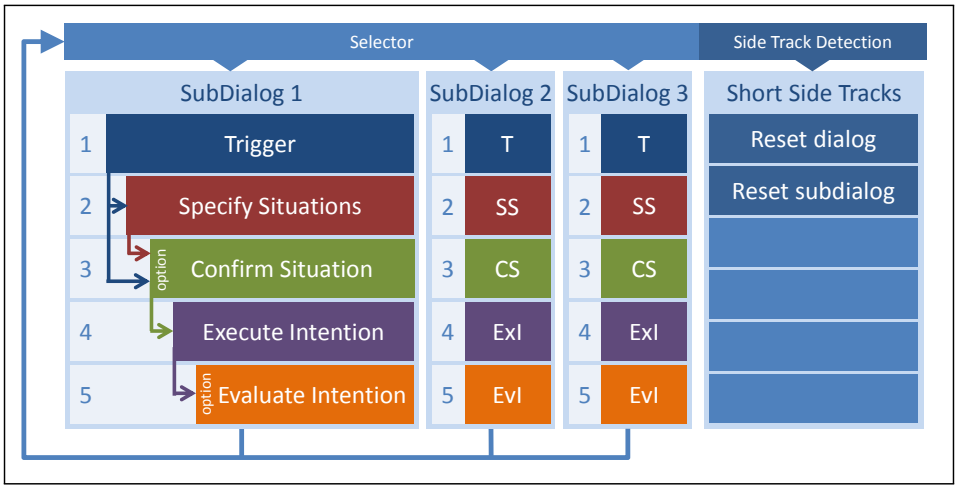
5 Concept en dialoogdata

De analyse heeft als input gediend voor het ontwikkelde concept. De onderdelen van dit concept worden in dit hoofdstuk uiteengezet en vertaald naar naar het systeem. Aansluitend wordt het proces van dialoog modellering beschreven. Dit proces legt de link tussen een dialoog en het concept. Daarnaast geeft het een overzicht van de gebruikte terminologie, waarmee de data componenten in beeld worden gebacht.

5.1 Het concept

In het hoofdstuk analyse is er een dialoogmodel opgesteld dat de bouwstenen van een dialoog definïeerd. Dit model is vertaald naar een concept voor dialoogopbouw, afhandeling en implementatie. in figuur 13 wordt het concept visueel weergegeven, gevolgt door een korte beschrijving van de onderdelen.

Figuur 13: Visuele weergave van het concept



1. Ten eerste heeft elke subdialoog een *trigger*. Deze zorgt voor het selecteren van de subdialoog. Dit wordt opgebouwd uit een regel in een *grammarfile*, 'selector' genaamd.
2. Als de subdialoog actief is geworden, komt die in stap twee waar de situatie wordt onderzocht. Deze bestaat uit *information needs* vanuit prolog. Het kan zijn dat er geen *information needs* actief zijn, omdat deze bijvoorbeeld al eerder zijn voorzien van informatie of ze zijn al voorzien bij de *triggering* van de subdialoog. In dat geval zal het systeem stap twee automatisch overslaan
3. Stap drie is een optionele stap. Deze stap zorgt voor een extra bevestiging van de situatie. Met een output als 'Klopt het dat ..' kan deze stap gebruikt worden voor het inbouwen van extra bevestiging. Het wordt voor de gebruiker mogelijk deze in de interface aan/uit te zetten.
4. Stap vier is de uiteindelijke uitvoer van de intentie van de subdialoog. Dit kan zijn: voorzien in de informatie behoefte door procedures te vertellen, informatie op te zoeken in externe databases, het uitvoeren van acties zoals het aanzetten van een lamp etc.
5. Stap vijf is net als stap drie optioneel. Deze geeft de mogelijkheid het systeem te laten vragen om een bevestiging / evaluatie van de output. De uitkomst hiervan kan resulteren in het activeren van een andere subdialoog om bijvoorbeeld veranderingen aan te brengen in een database.

5.2 De Selector

De selector is niets meer dan een *grammar* bestand dat de triggering van van de subdialog faciliteerd. In dit bestand worden de woord combinatie toegevoegd waar de spraakherkenner naar moet luisteren. De woord combinaties zijn gekoppeld aan *FIPA* berichten. Wanneer een uitspraak herkend is zal het gekoppelde bericht verstuurd worden naar het dialoogsysteem. Dit bericht dient als *trigger* voor een bepaald subdialoog of een 'short side track'. De selector wordt pas weer actief na het verlaten van een subdialoog. Een subdialoog heeft zijn eigen *grammar* files.

5.3 De Subdialoog

In het systeem zijn twee type subdialogen te onderscheiden, de twee typen laten zich typeren door de *speechacts* waar zij door *getriggerd* worden. Het eerste type laat zich *triggeren* door de *speechacts* 'Inform' of 'Request' en de ander door een 'Query-if'. De afhandeling van het eerste type bestaat uit het stellen van een serie vragen waarvan de antwoorden leiden tot een situatie. De situatie is weer gekoppeld aan een intentie, bijvoorbeeld het vertellen van een procedure of het uitvoeren van een actie of meerdere acties. De *query-if* wordt voornamelijk gebruikt voor het controleren of een feit waar of niet waar is, bijvoorbeeld; heb je de 'procedure begrepen?' of 'zijn er nog vragen?'

5.3.1 Request en Inform

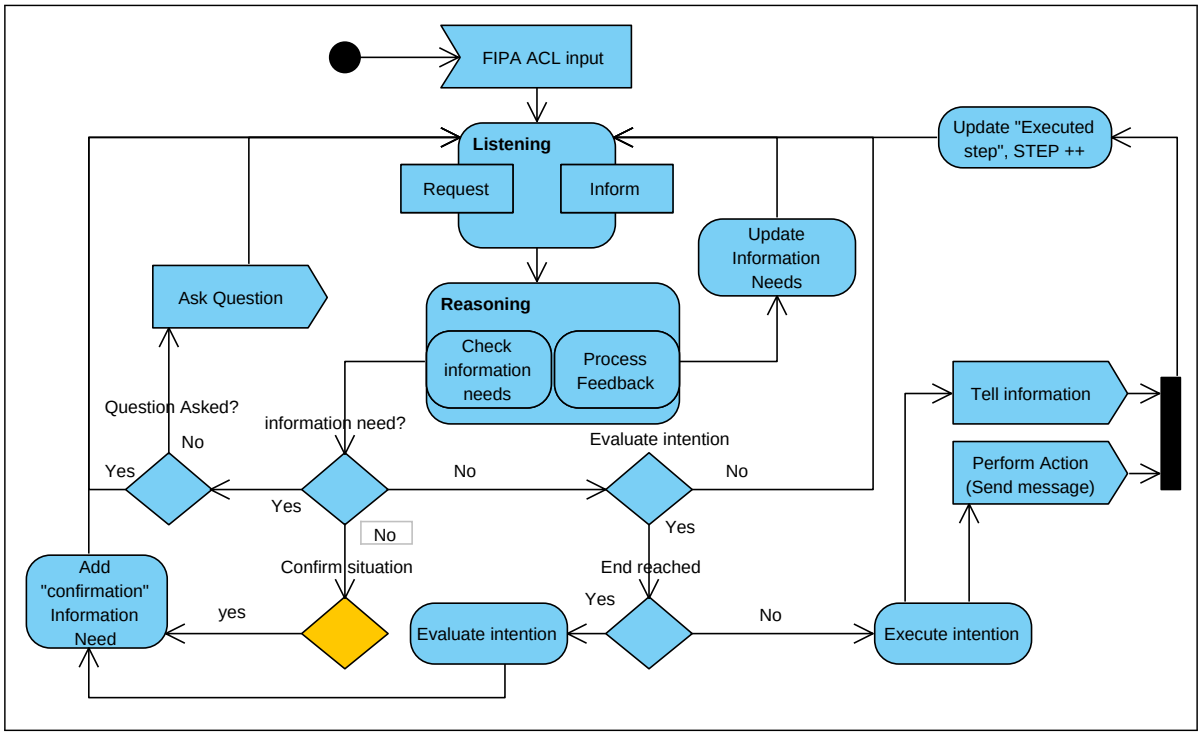
Het globale verwerking proces bij het ontvangen van een 'Inform' of een 'Request' *trigger* wordt gedemonstreerd met behulp van het scenario voor de hoofdpijn diagnose. Van dit scenario is een fractie te zien in tabel 8. Het volledige tabel is te vinden in bijlage D. De middelste kollom van het tabel beschrijft de verwerking van de dialoog (de redenatie). Het scenario met de input, de output en de redenatie stappen laten zich direct vertalen in het activiteiten diagram in figuur 14.

Table 8: Mogelijk verloop hoofdpijn diagnose scenario

User	**Reasoning**	**Ashley**
Ashley I have a headache!	Received *trigger*: Inform subdialog diagnose_headache	
	Add_belief: current_subdialog = diagnose_headache	
	Check information needs = attack_duration	
	Determine corresponding question	Do the attacks last more than 4 hours or less?
I guess they last more than 4 hours	Process the Feedback / Add belief: Attack_duration = more_then_4_hours / Update information needs	
	Check *information needs* = Type of turn	
	Determine corresponding question	Can you describe your pain level in term of high, medium or low?
I think it is a medium pain	Process the Feedback / Add belief: level = medium / Update *information needs*	

De algemene redenatie loop voor de verwerking van een subdialoog van dit type, is te zien in het activiteiten diagram. De gesprekspartner van Ashley geeft aan dat hij of zij hoofdpijn heeft. De spraakherkenner zet de herkende spraak om in een *FIPA* bericht. Het bericht dient als *trigger* die de subdialoog initïeerd.

Figuur 14: Het verwerkingsproces van de Inform en de Request trigger



Tabel 9 laat de *FIPA* berichten zien. Opvallend is het bericht aan de rechter kant, dit is een combinatie bericht die twee dingen doet; het *triggeren* van een subdialoog en het direct invullen van de eerste informatie behoefte. Het kan handig zijn om deze constructie te gebruiken om een dialoog natuurlijker te laten verlopen. De dialoog zou niet natuurlijk overkomen als bepaalde informatie al is verteld en Ashley vraagt er vervolgens weer om.

Table 9: De opbouw van een Inform of Request trigger

Inform	**Request**	**Combinatie trigger**

`<inform>`
`  <subdialog>`
`  headache_diagnose`
`  </subdialog>`
`</inform>`
	`<request>`
	`  <subdialog>`
	`  headache_diagnose`
	`  </subdialog>`
	`</request>`
		`<request>`
		`  <subdialog>`
		`  headache_diagnose`
		`  </subdialog>`
		`  <attack_duration>`
		`  more_then_4_hours`
		`  </attack_duration>`
		`</request>`

Nu de subdialoog actief is zal Ashley haar kennis bijwerken door de huidige subdialoognaam op te slaan. Daarna zal Ashley beredeneren dat zij een informatie behoefte heeft en daar de vraag 'do the attacks last 4 hours or less?' voor genereren. De gesprekspartner zal daarop antwoorden met 'I guess they last more than 4 hours'. De spraakherkenner zal de herkende spraak dan weer omzetten in een *FIPA* bericht en deze naar het dialoogsysteem versturen (tabel 10).

Table 10: De opbouw van een Inform bericht met voorbeeld

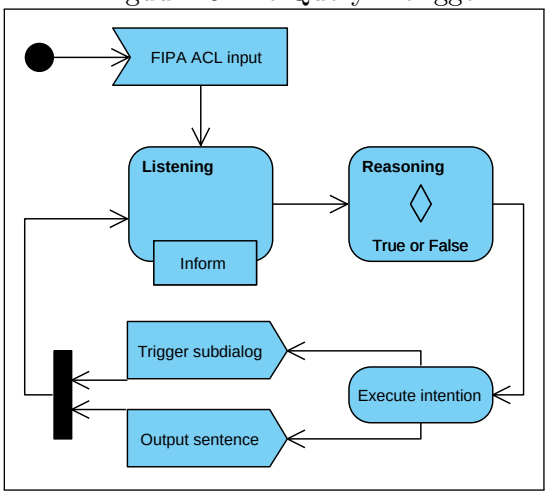
Opbouw	**Voorbeeld**
<inform> <situation> <info_need_content> answer_utterence </info_need_content> </situation> </inform>	<inform> <situation> <attack_duration> more_than_4_hours </attack_duration> </situation> </inform>

De feedback wordt vervolgens verwerkt en de kennis wordt bijgewerkt. Dit gaat zo door tot er geen *informatie needs* meer zijn. Optioneel is het bevestigen van de situatie (De gele diamant), de *information need* wordt toegevoegd en er wordt een vraag gesteld. Wanneer het antwoord hierop 'niet correct' is zal de subdialoog opnieuw beginnen. In het andere geval is Ashley nu klaar om de intentie uit te voeren. Een intentie bestaat uit 0 tot N stappen, dit kunnen output zinnen zijn of *FIPA* berichten die intern of extern gestuurd kunnen worden (Meer hier over in hoofdstuk 6.1.3). Als laatste kan er geïnventariseerd worden of de intentie goed is uitgoevoerd. De vraag en de actie die volgt kan de ontwerper zelf definieren.

5.3.2 Query-if

Deze trigger kan op twee manieren gebruikt worden, namelijk: om te controleren of iets begrepen is of om te controleren of een bepaald feit bestaat, dat wil zeggen of iets waar is.

Figuur 15: De Query-if trigger



Het activiteiten diagram (links weergegeven), geeft globaal het proces weer van de dataverwerking bij het ontvangen van een Query-if trigger. De spraakherkenner zou kunnen luisteren naar vragen als: Heb je de procedure begrepen? Heb je nog vragen? Is het lekker weer vandaag? Wordt jij Ashley genoemd? Tabel 11 laat een overzicht zien van de mogelijke *triggers*.
De dialoogontwerper bepaald de uit te voeren intentie die volgt wanneer de subdialoog is geïnitieerd. Bij zowel een waar of onwaar kan er een output zin worden gegenereerd of er kan een intern *FIPA-ACL* bericht worden verstuurd dat als *trigger* dient voor het inïtieren van een ander subdialoog.

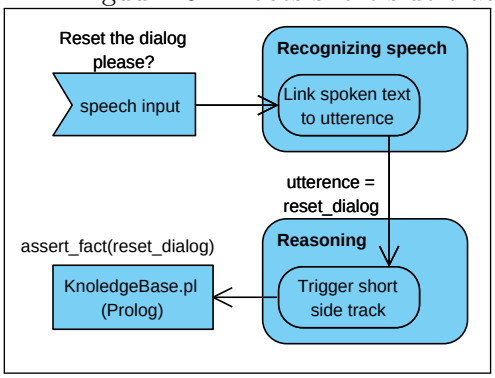
Table 11: Een overzicht van de Query-if trigger berichten

<queryif> <subdialog> subdialoog_naam </subdialog> <understood> is_fact(fact) </understood> </queryif>	<queryif> <subdialog> subdialoog_naam </subdialog> <understood> subdialog </understood> </queryif>	<queryif> <subdialog> subdialoog_naam </subdialog> <understood> </understood> </queryif>

5.4 Short side track

Een *short side track* is een 'klein' subdialoog, bestaand uit een *trigger* en een intentie. De short side tracks kunnen bijvoorbeeld gebruikt worden voor systeem acties als een reset van het gehele dialoog of alleen de huidige subdialoog.

Figuur 16: Proces short side track



De short side tracks dienen als hulpmiddel om terug te keren wanneer er problemen in het dialoog optreden. Dit is mogelijk omdat ze altijd *getriggerd* kunnen worden, ze zijn naast de uitvoer van een subdialoog altijd actief.
Op dit moment zijn er twee genoemde short side track geïmplementeerd; reset dialoog en reset sub_dialoog. Dat wil zeggen dat er een constructie aan de *Prolog* database is toegevoeg waarmee je de genoemde acties kunt uitvoeren. De woordcombinaties waar de spraakherkenningserver naar luisterd, kunnen aangepast worden in een user interface of in het XML bestand.

Afbeelding 16 geeft het verwerkingsproces van de short side track weer. Vanuit de spraakherkenner wordt er een *FIPA* bericht gekoppeld aan de herkende spraak . De *utterence* die hieraan gekoppeld is, is een directe verwijzing naar een *Prolog* regel. De *Prolog* regels voor de eerder genoemde resets zijn aangemaakt in de *Prolog* database. Het onderstaande voorbeeld laat het verwerkingsproces zien.

5.5 Ontwerpen en modeleren van subdialogen

Met behulp van een aantal dialoog voorbeelden wordt de link gelegd tussen het ontwerpen en aanmaken van een dialoog volgens het ontwikkelde concept. Het eerste scenario beschrijft een dialoog waarin Ashley een hoofdpijn diagnose stelt. Het opvolgende scenario beschrijft het 'Lost Wingman' scenario waar Ashley een aantal vragen stelt om de situatie te bepalen, waarna de procedure wordt verteld die hoort bij de situatie.

Beide voorbeelden zijn bedoeld om een beeld te vormen van het ontwerp proces van de dialoog. Daarnaast demonstreren de voorbeelden wat het concept betekent voor het ontwerpproces en welke data van belang is.

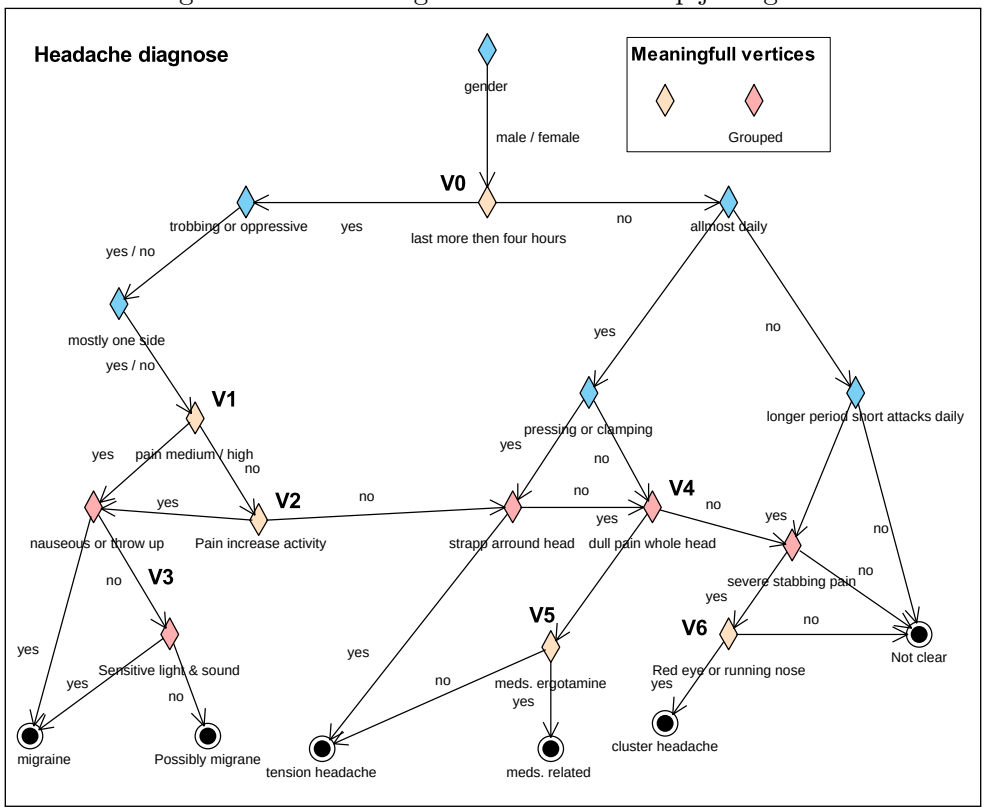
5.5.1 Hoekpunten (Vertices)

In de oude situatie werdt de transitie naar een volgende hoekpunt bepaald door alle voorgaande antwoorden. In plaats van de voorgaande antwoorden te onthouden, onthoud het systeem nu de vorige hoekpunt. Een hoekpunt is een preconditie van een volgend hoekpunt of situatie. Het grootste voordeel van deze methode is, dat het systeem af kan met het vervullen van één *information need* per vraag en er geen complexe constructies voor het zetten van *information needs* bedacht hoeven te worden. Een voorbeeld hiervan is bijvoorbeeld een symptoom. Een diagnose bestaat vaak uit meerdere symptomen. Het kan dus voorkomen dat er met één antwoord soms meerdere symptomen moeten worden toegevoegd en soms is het juist belangrijk te onthouden dat een symptoom niet is opgetreden. Doordat het vorige hoekpunt een voorwaarde is voor een nieuw hoekpunt en/of eindsituatie, is het niet nodig deze complexe constructies te maken. Daarnaast is de kennis over het huidige hoekpunt van belang voor het selecteren van de juiste outputzin, de output zinnen worden gelabeld met de naam van het hoekpunt. Het kan namelijk voorkomen dat een information need in een andere context, andere een andere antwoord/vraag combinaties heeft.

5.5.2 Hoofdpijn diagnose

Figuur 17 laat een gerichte graaf zien voor het uitvoeren van een diagnose conversatie, de diagnose wordt gebruikt om de aard van de hoofdpijn te bepalen[2]. In dit voorbeeld worden er vragen gesteld die met 'Ja' en 'Nee' beantwoord kunnen worden. De vragen leiden tot een van de eindsituaties: migraine, mogelijk migraine, spanningshoofdpijn, clusterhoofdpijn en niet duidelijk.

Figuur 17: Gerichte graaf voor een hoofdpijndiagnose



Informatie behoeften bepalen

Om deze informatie om te kunnen zetten in een subdialoog moeten de kenmerken (feiten) die leiden tot een situatie worden bepaald. Een situatie in dit geval is de gestelde diagnose, dus bijvoorbeeld migraine of spanningshoofdpijn. Een kenmerk of (feit) is een gegeven behorende tot een bepaalde situatie, bijvoorbeeld een symptoom zoals kloppende pijn of misselijkheid. Door deze gegevens te clusteren worden de mogelijke informatie behoeften van de subdialoog bepaald.

Table 12: Geclusterde kenmerken vormen de informatie behoeften

symptom	trobbing, opressive, increase physical activity, nauseous, throw-up, sensitive light,sensitive sound, dull pain, strapp around head, pressing, clamping, red eye, runny nose, stabbing pain
pain_area	one side, whole head
pain_level	medium, high, short attacks
attack_duration	longer than 4 hours, less than 4 hours
interval	almost daily, longer periods
external_influence	meds with ergotamine

[2]Afgeleid van de online hoofdpijntest

Hoekpunten definiëren

De gerichte graaf voor de diagnose in afbeelding 17 bestaat uit een serie 'Ja' of 'Nee' vragen, dit is een ideale manier voor een interface zoals een website. Voor een interface zoals Ashley kan dit op een andere manier gedaan worden, namelijk door het bepalen van de minimale condities voor een eindsituatie. De minimale condities voor migraine zijn bijvoorbeeld 'last more the 4 hours', 'pain medium/high of increase during activity' en 'nauseous of sensitive'. Daarna kan er bepaald worden welke condities van belang zijn om eindsituaties uit te sluiten, dit zijn de hoekpunten die er toe doen. Bepaalde hoekpunten kunnen gecombineerd worden in één vraag, bijvoorbeeld 'Can you describe your headache as a strapp arround your head, a dull pain in the whole head, a severe stabbing pain or neither'. Deze aanpak stimuleert een meer natuurlijk dialoog. In figuur 17 is gemarkeerd welke relevante hoekpunten er zijn en welke gegroepeerd zijn. Dit heeft geresulteerd in een diagnose met een set combinaties bestaande uit de onderstaande 7 globale vragen.

1. *V0: Do the attacks last more than 4 hours or less?*
2. *V1: Describe your pain level in term of high, medium or low?*
3. *V2: Does the pain increase during physical activity?*
4. *V3: Did you have any of the symptoms nauseousness, sensitive to light and sound or did throw up?*
5. *V4: Can you describe your headache as a strapp arround your head, a dull pain in the whole head, a severe stabbing pain or neither?*
6. *V5: Are you using meds that include the chemical ergotamine?*
7. *V6: Did you notice any red eyes or did you have a running nose.?*

De vragen kunnen meerdere antwoorden hebben, de dialoogontwerper bepaalt zelf naar welke antwoorden er geluisterd wordt en welke *utterences* daar aan gekoppeld worden. In tabel 13 wordt de relatie tussen de hoekpunten en de *utterences* aangegeven. Daarnaast is te zien welke *information needs* actief zijn/worden bij het betreden van het hoekpunt.

Table 13: Vertex definitie 'headache diagnose' met actieve information need(s)

Vertex defenitie	**Utterences**	**Information need**
$V0 :< A, B >$	A = more, B = less	attack_duration
$V1 :< C, D, E >$	C = high, D = medium, E = low	pain_level
$V2 :< F, G >$	F = increase_activity, G = neither	symptom
$V3 :< H, I, J >$	H = nauseous, I = sensitive, J = neither	symptom
$V4 :< K, L, M, N >$	K = strapp, L = dull, M = stab, N = neither	symptom
$V5 :< O, P >$	O = meds_ergotamine, P = not_meds_ergotamine	external_influence
$V6 :< Q, R, S >$	Q = red eye, R = running nose, S = neither	symptom

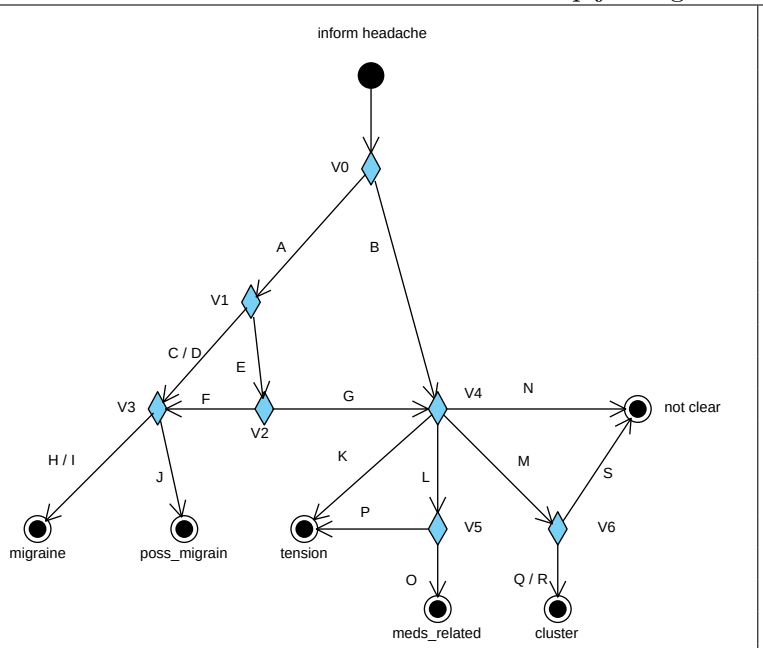
Opmerking: Bepaalde gevallen hebben een neither utterence. Dit vertaald zich in meerdere utterences die die ontkennend van aard zijn, bijvoorbeeld; not_nauseous en not_sensitive. Om de juiste transitie te maken is er maar één ontkennend feit nodig.

Transities bepalen

Een transitie van hoekpunt naar hoekpunt, of eindsituatie, wordt bepaald door het antwoord op de vraag. Afbeelding 14 is de gemodelleerde versie van de hoofdpijn diagnose in afbeelding 17 en geeft de transities visueel weer met daarnaast de logische representaties.

Een logische representatie als $(V0 \bigwedge A) \Rightarrow V1$, vertaald zich naar; als we in hoekpunt 'V0' zitten en het antwoord op de vraag is 'A' dan wordt het volgende hoekpunt V1.

Table 14: Gemodelleerde hoofdpijn diagnose met transities



$(V0 \bigwedge A) \Rightarrow V1$
$(V1 \bigwedge B) \Rightarrow V2$
$(V1 \bigwedge (C \bigwedge D)) \Rightarrow V3$
$(V2 \bigwedge F) \Rightarrow V3$
$(V0 \bigwedge B) \Rightarrow V4$
$(V2 \bigwedge G) \Rightarrow V4$
$(V4 \bigwedge L) \Rightarrow V5$
$(V4 \bigwedge M) \Rightarrow V6$
$(V3 \bigwedge (H \bigvee I)) \Rightarrow migraine$
$(V3 \bigwedge J) \Rightarrow possmigraine$
$(V4 \bigwedge K) \Rightarrow tension$
$(V5 \bigwedge P) \Rightarrow tension$
$(V5 \bigwedge O) \Rightarrow medsrelated$
$(V6 \bigwedge (Q \bigvee R)) \Rightarrow cluster$
$(V4 \bigwedge N) \Rightarrow notclear$
$(V6 \bigwedge S) \Rightarrow notclear$

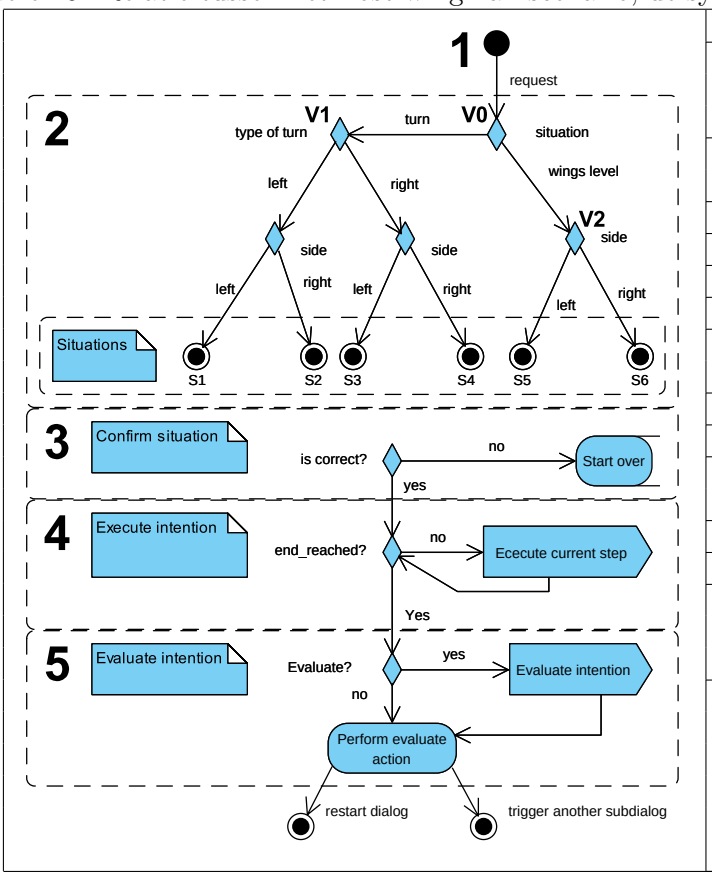
5.5.3 Lost Wingman

Dit is een eenvoudig subdialoog waarin Ashley gevraagd wordt een bepaalde procedure te vertellen. Door een serie vragen en antwoorden wordt de situatie bepaald en wordt uiteindelijk de procedure verteld. Tabel 16 laat de graaf zien van de mogelijke verlopen van dit scenario. De afbeelding illustreert hoe het scenario in het vijfstappenplan van het concept past en hoe deze afgehandeld wordt door het dialoogsysteem(Figuur 14). Aan de rechter zijde is een mogelijk conversatie verloop beschreven, de 'U' staat voor user en 'A' voor Ashley.

Table 15: Gemodelleerd 'Lost Wingman' scenario

Vertex met outputzin			**Transities**
V0: Are we in a turn or flying wings-level?			$(V0 \bigwedge A) \Rightarrow V1$
V1: What type of turn?			$(V0 \bigwedge B) \Rightarrow V2$
On which side am I flying?			$(V1 \bigwedge C \bigwedge E) \Rightarrow S1$
V2: On which side am I flying?			$(V1 \bigwedge C \bigwedge F) \Rightarrow S2$
Vertex defenitie	**Utterences**	**Information need**	$(V1 \bigwedge D \bigwedge E) \Rightarrow S3$
$V0 :< A, B >$	A = turn, B = wings_level	type_of_turn, side	$(V1 \bigwedge D \bigwedge F) \Rightarrow S4$
$V1 : (< C, D > \bigwedge < E, F >)$	C/E = left, D/F = right	pain_level	$(V2 \bigwedge G) \Rightarrow S5$
$V2 :< G, H >$	G = left, H = right	side	$(V2 \bigwedge H) \Rightarrow S6$

Table 16: Relatie tussen het Lost wingman scenario, de systeemverwerking en het concept



	User / Ashley
1	U: Ashley can you tell me the lost wingman procedure?
2	A: Are we in a turn or flying wings level?
2	U: We are in a turn
2	A: What type of turn?
2	U: A right turn?
2	A: What side am I flying?
2	U: You are flying on the right side
3	A: Is this correct?
3	U: Yes that is correct
4	A: This is step N of intention
5	A: Is this correct?
5	U: Yes it is
5	A: Send internal FIPA message Reset or trigger other subdialog.
5	A: Okay!

Tussen het model voor de diagnose en het model voor de Lost Wingman zit een wezenlijk verschil. In het model voor de diagnose behoort er één vraag of *information need* tot een hoekpunt. In het model voor de Lost Wingman is te zien dat hoekpunt V1 bestaat uit twee *information needs* met ieder zijn bijbehorende vraag en mogelijke antwoorden. Logischerwijs zou je zeggen dat de hoekpunten met de label 'side' gegroepeerd kunnen worden onder V2, dit gaat helaas niet op omdat dan voor alle eindsituaties $\forall S : (V2 \bigwedge (G \bigvee H))$ geldt als preconditie. Dit betekend dat er geen onderscheid meer gemaakt kan worden in de eindsituatie. Daarom is er gekozen om een uitzondering te maken op de definitie van de term 'hoekpunt'.

6 Resultaten

In hoofdstuk 1 en 2 zijn de problemen en wensen geïnventariseerd, met deze inventarisatie is een visie gevormd. De visie beschrijft een zogenaamde 'dialoog generator' waarmee je op een eenvoudige manier een dialoog kunt genereren. Om deze visie te verwezenlijken is er veel onderzocht. Onderdeel van het onderzoek was een analyse van de componenten die benodigd zijn om een dialoog te implementeren en uit te voeren. Overige wensen zoals het koppelen van externe modules en een zo generiek mogelijk systeem zijn hierin meegenomen. Daarnaast is er literatuur gelezen op het gebied van dialoogsystemen en dialoog modellen. De resultaten van de analyse en het literatuur onderzoek hebben geleid tot het ontwikkelde concept.

Dit hoofdstuk begint met het beschrijven van het concept met alle bijhorende resultaten. Daarna volgt de dialoog generator. De aanpassingen die hebben geleid tot een generiek systeem worden beschreven, dit is een resultaat op zich. Tot slot wordt er gekeken naar de resultaten voor het verbeteren en versnellen van het testproces.

6.1 Het concept

Eén van de resultaten is het ontwikkelde concept, het concept beschrijft een vijfstappenplan. Uit de analyse in hoofdstuk 4 is gebleken dat elk willekeurig dialoog - dat wordt beschreven in dit document - binnen de kaders van dit model te plaatsen is (Zie hoofdstuk 4 en 5. Om een conversatie te modelleren in een dialoog volgens het concept, is er een duidelijk beeld nodig van alle elementen die hiertoe behoren en hoe deze samenhangen met elkaar. Dit is gerealiseerd door een aantal voorbeelden te schetsen waarin het proces van de dialoog modellering gedemonstreerd wordt (zie paragraaf 5.5). Naast de visuele modellen is er ook een terminologie opgesteld dat de componenten van de dialogen beschrijft.

Het concept is ontwikkeld als standaard voor dialoog implementatie. De kern van het concept bestaat uit het vijfstappenmodel. Naast het vijfstappenmodel is het van belang te beschrijven wat dit concept betekend voor de interactie tussen subdialogen en de koppeling met externe modules. Deze resultaten brengen vele mogelijkheden voor het systeem. Als bijkomstig resultaat wordt de optimalisatie van de spraakherkenning beschreven.

6.1.1 Interactie tussen subdialogen

De invulling van de dialoog met subdialogen is geheel te bepalen door de dialoogontwerper. Een dialoog kan opgebouwd worden uit een groep subdialogen die niet hetzelfde onderwerp delen. Deze aanpak maakt het mogelijk om veel kleine scenarios snel uit te voeren en te testen. Een voorbeeld van deze aanpak is de combinatie 'Apparatenhulp' en 'Lost Wingman'. Deze twee subdialogen hebben geen enkele overeenkomst en staan volledig los van elkaar. Er kan ook gekozen worden een complex dialoog op te bouwen uit verschillende subdialogen door middel van 'Subdialog Chaining'(paragraaf).

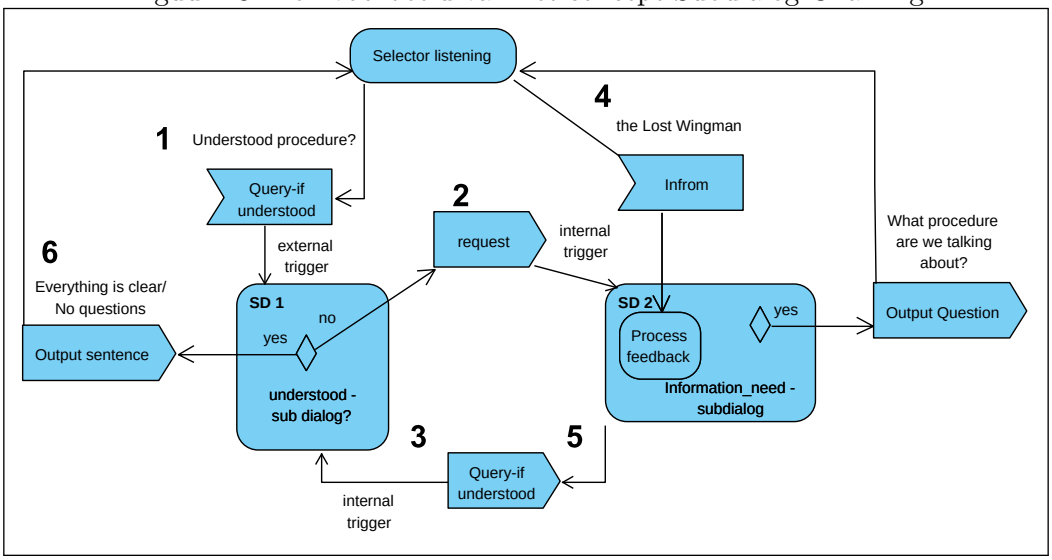
Subdialogen kunnen intern en extern geactiveerd worden. Als een subdialoog een ander subdialoog activeert, wordt dit interne *triggering* genoemd. Externe *triggering* van een subdialoog kan alleen wanneer de selector actief is. De selector is gedeactiveerd wanneer een subdialoog actief is en wordt weer actief na het verlaten van een subdialoog. De *grammar* voor short side tracks is wel altijd actief.

Om informatie te delen tussen subdialogen kan een feit (een combinatie van een *information need* met een *utterence* als invulling) gedefinieerd worden als 'global knowledge'. Dit houdt in dat de informatie niet verloren gaat bij het verlaten van een subdialoog. De global knowledge blijft bewaard tot de dialoog (Alle subdialogen gecombineerd) een reset krijgt.

6.1.2 Subdialog Chaining

Erg belangrijk is het bedachte 'Subdialog Chaining'. Dit resultaat stimuleert een natuurlijk dialoog verloop. Hoe dit in zijn werk gaat is het best te demonstreren met het voorbeeld weergegeven in figuur 18.

Figuur 18: Een voorbeeld van het concept Subdialog Chaining



In de weergave zijn de stappen genummerd van 1 tot 6. Er zijn twee subdialogen te zien SD1 en SD2. Er zijn twee activiteiten: listening en process feedback. De subdialogen ontvangen externe berichten, versturen interne berichten en produceren output zinnen.

Het voorbeeld geeft een aantal interacties weer tussen subdialoog 1 en 2. Te zien is de 'Selector', deze luistert naar de gedefinieerde woordcombinaties en zal daar een *trigger* aan koppelen. De genummerde stappen worden hieronder nader toegelicht.

1. Er komt een *trigger* binnen van de spraakherkenner. Dan wordt er gekeken of het gevraagde begrepen is. Het gevraagde is niet begrepen wanneer er geen subdialoog voor een bepaald onderwerp actief is. Het kan ook zij dat deze wel actief is, maar dan met actieve informatie behoeften.
2. Wanneer het gevraagde niet begrepen is, dan wordt subdialoog 2 geactiveerd door een intern bericht (*trigger*).
3. Subdialoog 2 bevat de structuur voor het verkrijgen van informatie over de subdialoog en formuleert daar een vraag of meerder vragen voor.
4. Er komt een bericht binnen afkomstig van de spraakherkenner met daarin de herkende response, dit antwoord wordt verwerkt.
5. Subdialoog 2 heeft de intentie subdialoog 1 te activeren (*trigger*).
6. Subdialoog 1 heeft nu dankzij subdialoog 2 het gevraagde begrepen en genereert de output zin 'Everything is clear'.

6.1.3 Koppeling met externe modules

Het dialoogsysteem van Ashley is een onderdeel van een gedistribueerd systeem. Dit betekent dat er een speciaal ingerichte module aan het geheel toegevoegd kan worden. De taak van de module zou kunnen zijn, het aanzetten, uitzetten of dimmen van een lamp.

De dialoogontwerper kan op meerdere manieren *FIPA* berichten definiëren, dit kan bij een 'Confirm Situation', de 'Execute intention' en de 'Evaluate intention'. Er kan dus een bericht worden verstuurd naar een externe module, de module begrijpt het bericht en voert de actie uit, de module schakelt het licht aan en stuurt een bevestiging terug naar het dialoogsysteem, deze bevestiging kan weer dienen als *trigger* voor een eventueel ander subdialoog.

In de beschrijvingen van Ashley wordt het AI component beschreven als het brein van Ashley. Naast de redenering over de dialoog, kan Ashley externe modules aansturen met *FIPA* berichten. Andersom kunnen externe modules ook berichten naar Ashley sturen waarmee een subdialoog getriggerd kan worden. Het brein van Ashley kan dankzij dit resultaat, gekoppeld worden aan zintuigen (sensoren) en beschikt nu ook over motorische mogelijkheden.

6.1.4 Optimalisatie spraakherkenning

Een van de beperkingen van spraakherkenningsystemen is dat spraak wordt herkend in een bepaalde context. Een herkende uitspraak zoals 'hard' kan in verschillende contexten gebruikt worden, bijvoorbeeld: 'rijden we **hard** of zacht?' of 'Wil je een zacht of **hard** gekookt ei?'. Als er één *grammar* bestand zou bestaan voor beide situaties, zou 'hard' gekoppeld worden aan de context die het eerst voorkomt. Om dit te voorkomen mag een *grammar* bestand geen conflicterende woordcombinaties bevatten. De oplossing hiervoor is simpelweg, voor elke context een eigen *grammar* bestand maken en deze activeren wanneer nodig.

In de oude situatie werd er in het dialoogsysteem op diverse plekken een nieuw *grammar* bestand geactiveerd. Zo bestond er één voor het groeten van de gesprekspartner, voor het luisteren naar woordcombinaties die het Lost Wingman scenario activeren en één om de antwoorden te herkennen op vragen die door Ashley gesteld worden.

In de nieuwe situatie kunnen er vele *grammar* bestanden bestaan voor een dialoog. Er zal er altijd één bestaan voor de 'selector' die de *triggers* bevat voor het activeren van de subdialogen. Voor elk subdialoog bestaat er minimaal één *grammar* bestand waarin de woordcombinaties gespecificeerd waar naar geluisterd wordt in de subdialoog. Om de spraakherkenning te optimaliseren kan er voor elke vraag die gesteld wordt (door Ashley) een apart bestand worden aangemaakt. Dit kan bij de stappen specify situation, confirm situation en evaluate intention uit het vijfstappenplan. Deze optimalisatie geeft de ontwerper veel vrijheid en minimaliseert de kans op foutief herkende spraak.

6.2 Generalisatie

Om het concept op een goede manier te kunnen implementeren is het systeem generiek gemaakt. In beginsel stonden de opdrachtgevers hier nogal sceptisch tegenover. Door een aantal aanpassingen te maken is dit toch gelukt.

Om te beginnen moesten de componenten in kaart worden gebracht die specifieke data verwerken. Deze zaten voornamelijk in de opbouw van *proposities* en de uitvoer van bepaalde *goals*. Met behulp van *reverse engineering* is de logische werking van deze *goals* in kaart gebracht. In de bijlage is een klassendiagram te vinden die aangeeft - door middel van highlights - welke onderdelen specifieke data verwerken (zie bijlage C).

Eerder is het element 'Procedure' genoemd en apart geanalyseerd (zie paragraaf 4.3). Dit element leent zich uitstekend voor het ontwikkelde concept en is omgebouwd tot 'Intention'. Dit is gedaan om het element generiek te maken zodat het voor elk willekeurig scenario gebruikt kan worden. De 'Intention' zoals deze bestaat als element in het dialoogsysteem, zorgt voor de stapsgewijze afhandeling van een doel. Een stap van een doel kan een bepaalde uitspraak zijn of het versturen van een intern of extern bericht.

Een andere generalisatie slag is het aanbrengen van een goede scheiding tussen de kennis over de dialoog en de specifieke dialoogkennis. Voorheen werden er codeblokken toegevoegd in diverse

source bestanden voor elk geïmplementeerd scenario. Dit is nu veranderd, de verantwoordelijkheid over de kennis van de dialoog is aan *Prolog* overgelaten en in Java zijn de lange codeblokken vervangen door het raadplegen van de *Prolog* kennisbase. Deze aanpassing heeft honderden regels code overbodig gemaakt.

De *Prolog* kennisbase bestaan uit drie bestanden: de knowledgebase, sentences en messages. De eerste beschikt over een structuur voor het redeneren over de dialoog, dit bestand bevat ook de dialoogkennis. De output van het systeem is opgeslagen in het tweede en derde bestand. De output kan bijvoorbeeld een vraag zijn of het groeten van iemand. Dit zijn zinnen die worden gehaald uit het bestand 'sentences'. De output zou ook een bericht kunnen zijn naar een externe module, of een intern bericht voor het activeren van een bepaald onderdeel van de dialoog. Deze berichten worden opgeslagen in het bestand 'messages'.

De generalisatie heeft er toe geleid dat de Java code volledig generiek is en dat de dialoog generator de benodigde constructies kan genereren in de Prolog kennisbase.

6.3 De dialoog generator

In hoofdstuk 2 is de visie gevormd van de dialoog generator. De dialoog generator heeft als input: de source van het dialoogsysteem en een XML bestand dat een dialoog specificeert. De output is een gecompileerd dialoog systeem met alle bijhorende bestanden. De XML en de opzet van de dialoog generator worden hieronder nader beschreven.

Als verbindende laag tussen de gemodelleerde data, de dialoog generator en een eventuele gebruikers interface, is XML gebruikt. De dialoog opbouw is gespecificeerd in een zogenaamd *XSD* bestand dat zorgt voor validatie aan de hand van het vijfstappenplan in het concept. De gemodelleerde data kan handmatig in XML worden ingevoerd of mogelijk middels een gebruikers interface die het invoer proces begeleid. In bijlage E is een voorbeeld XML bestand toegevoegd dat de dialoogstructuur van een hoofdpijndiagnose bevat.

De generator draagt zorg voor de codegeneratie in *Prolog* en het aanmaken van *grammar* bestanden. Voor het genereren van deze data is de API ontwikkeld. De API bestaat uit een verzameling klassen waarvan de benamingen overeenkomen met de gebruikte terminologie van het concept. Het genereren van code en het aanmaken van *grammar* bestanden, gebeurt op basis van de dialoogspecificatie in het XML bestand. De API beschikt over methoden die het dialoogsysteem kunnen compileren en afleveren als overkoepelend demo bestand. Het klassendiagram van de API is met beschrijvingen te vinden in bijlage F.

Er is nu een tool ontstaan waarmee een dialoogontwerper zonder bijzondere technische kennis, gemakkelijk een dialoog kan implementeren, met behoud van de mogelijkheid tot aanpassen van de originele broncode.

6.4 Testen

Om het testproces te verbeteren en te versnellen is het van belang een aantal zaken te weten, namelijk: hoe wordt er op het moment getest, welke zijn de onderdelen die inzichtelijk gemaakt moeten worden en hoe kan ik die op een overzichtelijke manier presenteren.

In de oude situatie werdt het dialoogsysteem getest door te printen naar een log op het scherm of in een bestand. De informatie output van het printen is prima, het probleem is echter dat er heel veel informatie geprint wordt, dit maakt het zoeken naar de benodigde informatie een hele taak. Daarnaast blijft er tijdens een systeemrun output gegenereerd worden, waardoor je eigenlijk alleen het log kan doorzoeken wanneer het systeem gestopt is. Om dit probleem op te lossen wordt er geprint naar een socket, dat wil zeggen dat de log berichten over TCP/IP verstuurd worden naar een programma die deze kan verwerken. Dit programma kan dan de log berichten filteren en live weergeven terwijl het systeem operationeel is.

Om het dialoogsysteem te kunnen testen zonder spraakherkenner, bestond de optie om de dialoog input uit een bestand te lezen. Normaal gesproken stuurt het dialoog systeem *FIPA* berichten op, deze zijn gekoppeld aan de herkende spraak. Wanneer het dialoogsysteem getest wordt met input vanuit een bestand, dan moeten deze *FIPA* berichten handmatig in het bestand één voor één worden weggeschreven om te kijken of het dialoog systeem ze goed verwerkt. Dit kost veel

tijd en is zeer foutgevoelig, een kleine typefout kan al voor problemen zorgen. Het komt erop neer dat het systeem eigenlijk altijd *FIPA* berichten verwerkt. In testmodus komen de berichten binnen via een bestand en anders van de spraakherkenner. De oplossing voor dit probleem is het systeem aan te passen zodat er geen onderscheid bestaat tussen berichten uit het test bestand en van de spraakherkenner.

In een ontwikkeld dialoog kan het voorkomen dat er voor meerdere situaties dezelfde woord combinaties nodig zijn. Door in het verloop van een dialoog te schakelen tussen *grammar* bestanden, kan de herkende spraak in de juiste context geplaatst worden. Het is dus voor de tester van belang, te weten of er correct geschakeld wordt en of er nergens conflicten optreden. Zoals beschreven in paragraaf 3.1, is Ashley opgesteld in twee ruimtes. De spraakherkenningsserver staat achter de schermen en niet in de interactie ruimte. Welk *grammar* bestand actief is, is te zien tussen de log berichten en op de spraakherkenningsserver. Dit is niet optimaal en zorgt voor veel onderbrekingen in het testproces.

Het bovenstaande heeft betrekking op het verbeteren van de oude situatie, in de nieuwe situatie is het concept ontwikkeld. Het concept bestaat uit een vijfstappenplan, waarbij elke stap zijn eigen onderdelen heeft. Om het testen goed aan te laten sluiten op dit concept is het dus van belang om deze onderdelen goed inzichtelijk te maken.

De genoemde problemen en aanbevelingen, zijn verwerkt in een eenvoudige GUI. De GUI heeft een bidirectionele verbinding met het dialoogsysteem. Alle mogelijke *FIPA* berichten kunnen gedefinieerd worden in een bestand dat ingeladen wordt. In een weergavescherm van de GUI vindt je de *FIPA* berichten terug, door erop te klikken wordt het bericht verstuurd naar het dialoogsysteem. De log berichten zijn onderverdeelt in een aantal categorieën, te weten: debug, info, warn, error en fatal. De feedback weergave biedt een filter waarmee je de gewenste categorieën aan of uit kunt zetten.

7 Conclusies en aanbevelingen

Het ontwikkelingstraject is een uitdagende taak geweest. De onbekende terreinen op het gebeid van technologie, het doorgronden van een complex 'state of the art' systeem, het brainstormen over de mogelijkheden, het goed formuleren van de ideeën en oplossingen, de beperkte tijd. Gedurende het gehele ontwikkelingsproces is er continu gezocht naar een goede ballans tussen deze ingrediënten. Dankzij onder andere het flexibele karakter van de Scrum projectmanagement methode, waarmee je gefaseerde behapbare doelstelling kunt stellen en continu kunt bijsturen, is het resultaat een succes boven verwachting. Het succes heeft Ashley naar een hoger niveau gebracht.

Dankzij de 'Dialoog Generator' is Ashley toegankelijk gemaakt voor een dialoogontwerper zonder bijzondere technische kennis. De dialoogontwerper kan zijn dialoog modelleren volgens de beschreven modelleringmethode, deze methode vertaald de data naar het concept. Het dialoog ontwerp kan vervolgens gemakkelijk in XML ingevoerd worden. De ontwikkelde dialoog generator zal dan alle bijhorende onderdelen genereren in de code en een kant en klaar demo bestand opleveren.

Erg belangrijk is dat er nu eenvoudig een 'natuurlijk' dialoog geïmplementeerd kan worden, door 'Subdialog Chaining' toe te passen. In een 'natuurlijk' dialoog wordt er goed geanticipeerd op het conversatiegedrag van de gebruiker.

Voor het comminiceren met externe componenten, is de mogelijkheid voor het definieren van *FIPA* berichten toegevoegd aan Ashley. Deze kunnen verstuurd worden bij het uitvoeren van de 'intention', bij 'specify situation' en 'evaluate intention' van het vijfstappenplan. Hierdoor heeft Ashley ineens een motoriek, waarmee ze externe modules in het gedistribueerde systeem kan aansturen. Er kunnen nu ook diverse sensor modules gekoppeld worden die kunnen dienen als zintuigen.

Voordat het ontwikkelde dialoog in gebruik genomen kan worden, moet het getest worden. In de oude situatie was dit een tijdrovende taak omdat de belangrijke onderdelen en test data niet te overzien waren, deze konden achteraf pas goed geanalyseerd worden. Dit testproces is nu verbeterd door geen onderscheid te maken in het draaien met een spraakherkenner en het invoeren van de *FIPA* berichten in een bestand, het dialoogsysteem kent maar één invoer. Het systeem kan nu de testdata versturen naar een externe module, deeze module is een eenvoudige GUI die de data filtert en live laat zien. De data sluit goed aan bij het concept, het zorgt er voor dat je kunt volgen waar in het dialoog Ashley zich bevindt en waar zij op aan het wachten is.

7.1 Een mogelijk scenario in de nieuwe situatie

Ashley moet gereed gemaakt worden voor een nieuwe toepassing. De dialoogontwerper volgt de stappen van de dialoogmodellering. Zonder bijzondere technische kennis wil de dialoogontwerper, een complex dialoog implementeren voor het stellen van een diagnose ADHD. De dialoog moet zo natuurlijk mogelijk verlopen. Het concept 'Dialog Chaining' is gebruikt om te anticiperen op het conversatiegedrag van de gesprekspartner. Het dialoogmodel wordt omgezet in XML volgens het vijfstappenplan van het concept. De XML file is valide en de Dialoog generator compileert het geheel. Hij maakt de spraakherkenningbestanden aan en produceert een overkoepelend demo bestand. De dialoog ontwerper start een test waarbij hijzelf de patiënt is. Tijdens het testen zal hij de test interface gebruiken om het verloop van de dialoog te monitoren.

De patiënt komt de ruimte binnen, Ashley detecteerd dit en start het subdialoog voor het groeten. Ashley zegt: Welkom, waarmee kan ik u van dienst zijn. De patiënt verteld over zijn vermoedens ADHD te hebben. Ashley heeft dit begrepen start de subdialoog voor het stellen van een diagnose ADHD. Ze begint met het stellen van vragen. Aan het eind van de vragen ronde heeft ze de diagnose gesteld, deze vertelt zij aan de patiënt. Daarna vraagt ze of de patiënt akkoord gaat met een doorverwijzing naar de specialist. De patiënt is akkoord en Ashley activeert de subdialoog voor het plannen van een afspraak. De patiënt volgt een procedure voor het maken van een afspraak. De afspraak is gemaakt, Ashley plaatst deze in de agenda van de specialist en print vervolgens een

doorverwijzing uit met de afspraak details. Daarna activeert zij de dialoog voor het gedag zeggen van de patiënt.

7.2 Aanbevelingen

Ashley is een veelzijdig systeem en er zijn oneindig veel mogelijkheden te bedenken. Tijdens het ontwikkeltraject ontstaan er ideeën over mogelijke verbeteringen of nuttige toevoegingen. Helaas is een project tijdsgebonden en kan dit nooit allemaal meegenomen worden. De belangrijkste aanbevelingen worden hieronder kort beschreven.

7.2.1 Doorontwikkelen gebruikersinterface

In dit project is de ontwikkeling van een gebruikersinterface niet meegenomen. Er wordt een XML bestand aangemaakt en de dialoog generator verwerkt deze tot een demo bestand. De keuze voor XML, maakt het mogelijk vele interface types te gebruiken, bijvoorbeeld een webinterface, een GUI of eventueel Ashley zelf. Voor het testen is een eenvoudige test GUI gemaakt, deze zou verder doorontwikkeld kunnen worden.

7.2.2 Dialoogsynchronisatie

Het bovenstaande scenario beschrijft een ideaal conversatie verloop. Hierin gaat er niets mis. Stel dat er niet helemaal goed is geanticipeerd op het conversatiegedrag van de partner, de dialoog wordt onderbroken of er gaat iets mis met de koppeling van externe modules. Wat gebeurt er dan? Op dit moment zijn er twee mogelijkheden, je kunt het subdialoog resetten of je kunt het gehele dialoog resetten. Deze acties zijn uit te voeren door een 'short side track' aan te roepen. Naar de spraakcommando's voor de short side track wordt altijd geluisterd, ongeacht in welk subdialoog de dialoog zicht bevindt.

Dit is nogal basaal, er zouden meerdere short side tracks geïmplementeerd kunnen worden die bepaalde acties op het dialoog kunnen uitvoeren. Deze acties kunnen dan worden gebruikt door een subdialoog speciaal ontwikkeld voor het opvangen van problemen in de dialoog.

7.2.3 Dialooggeschiedenis

Een goede toevoeging aan Ashley zou kunnen zijn een dialoog geschiedenis bij te houden. De geschiedenis kan gebruikt worden voor de bovengenoemde synchronisatie doeleinden. Het zou een gesprek ook veel natuurlijker kunnen maken.

7.2.4 Groepsdialogen

De huidige Ashley is ontwikkeld om met één gesprekpartner te interacteren. Het is niet ondenkbaar dat Ashley ook in groepsverband een gesprek kan voeren, bijvoorbeeld bij lesgeven of een vergadering. Hiervoor zou de dialoog geschiedenis erg van pas komen. Ashley zal bij moeten houden met wie ze allemaal in gesprek is, de gesprekpartners kunnen onderscheiden en bijhouden wie wat heeft gezegd.

Begrippenlijst

Agile Letterlijk vertaald betekend dit lenig. In context slaat deze term op de projectmanagement aanpak. De agile methode is gebaseerd op iteratieve en incrementele ontwikkeling. 9

ASR 'Automatic Speech Recognition' is het omzetten van gesproken woorden naar tekst. 14

FIPA ACL FIPA 'Foundation for Intelligent Physical Agents' organisatie voor de bevordering van agent-based technologie. ACL 'Agent Communication language'. Een standaard taal voor onderlinge communicatie tussen agents. Deze standaard leent zich goed vanwege zijn achtergrond in autonome systemen en omdat het is gebaseerd op XML. 12

Goal Is een doel die de agent wil bereiken. 7

Goaltype Een Goaltype is een template klasse voor een bepaald type doel. 20

inputbelief Een veel gebruikte term op het gebied van autonome software. De term betekend: aannemen dat er informatie is ontvangen en deze opslaan in de kennis over de wereld.. 17

IVR 'Isolated Voice Reqognition' is een spraakherkenning methode waarbij de spreker geacht is te pauzeren tussen de gesproken woorden. 21

Prolog Prolog is een logische programmeertaal gebaseerd op predicatenlogica, de naam staat voor 'Programming in Logic'. In Prolog worden de voorwaarden waaraan een oplossing moet voldoen bepaald. In een programmeertaal zoals Java worden de stappen bepaald die naar de oplossing leiden. Prolog is zeer geschikt voor toepassingen met een kunstmatige intelligentie component en voor digitale verwerking van natuurlijke taal. 11

Propositie Een propositie is een voorstel die aangemaakt wordt door het systeem. De propositie is de basis voor het redeneerproces. 12

QA Question answer spraakherkenningsysteem. 21

reverse engineering Dit is het tegenovergestelde van: code generatie vanuit software modellen. Er worden dus modellen gegenereerd vanuit de code. 36

speechact Letterlijk vertaald betekend dit een taalhandeling. Een taalhandeling is bijvoorbeeld een bewering, belofte of een verklaring. 17

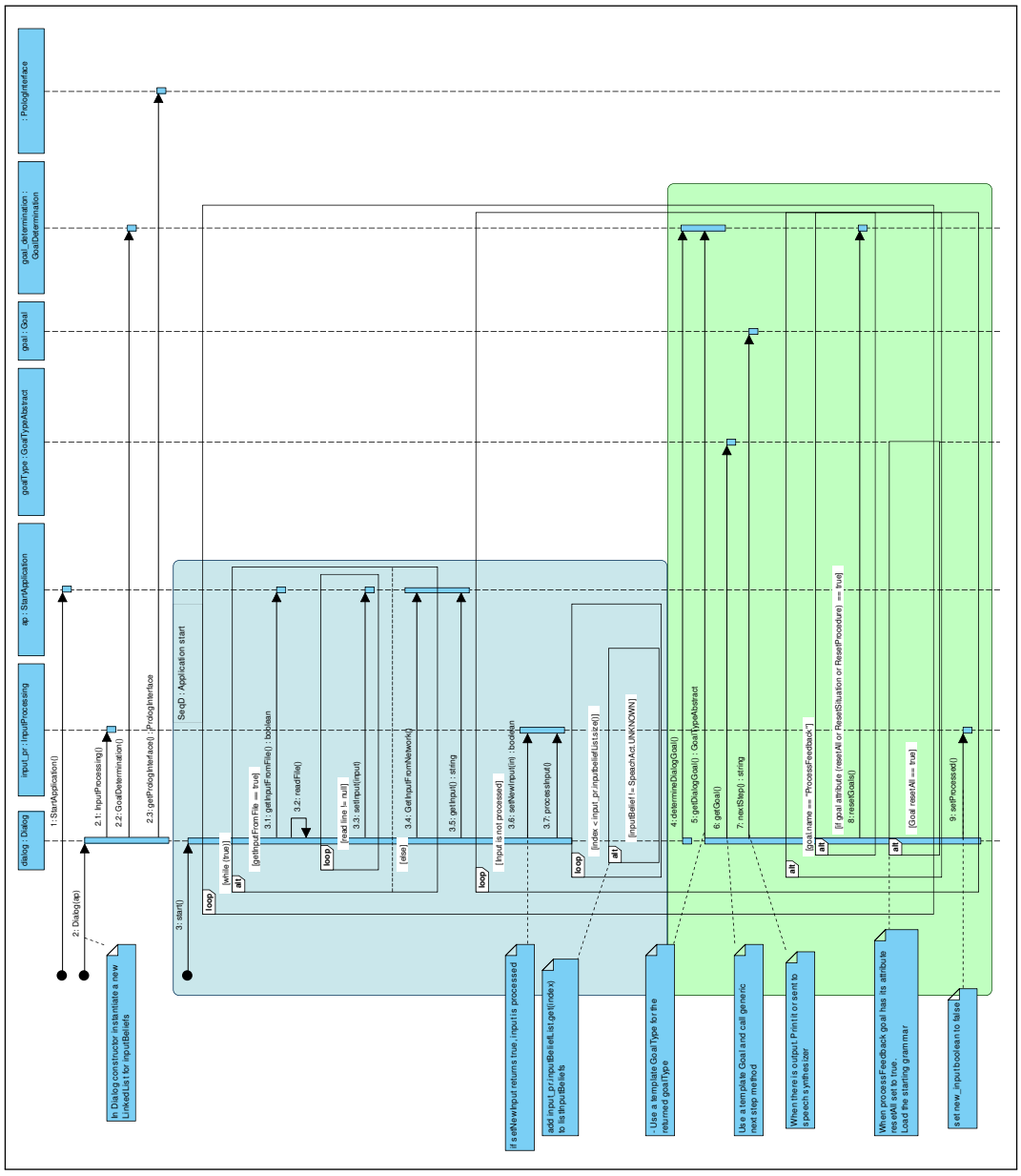
SRGS 'Speech Recognition Grammar Specification' is een W3C standaard voor het opstellen van woorden en woordpatronen, waar een spraakherkenner naar kan luisteren. 14

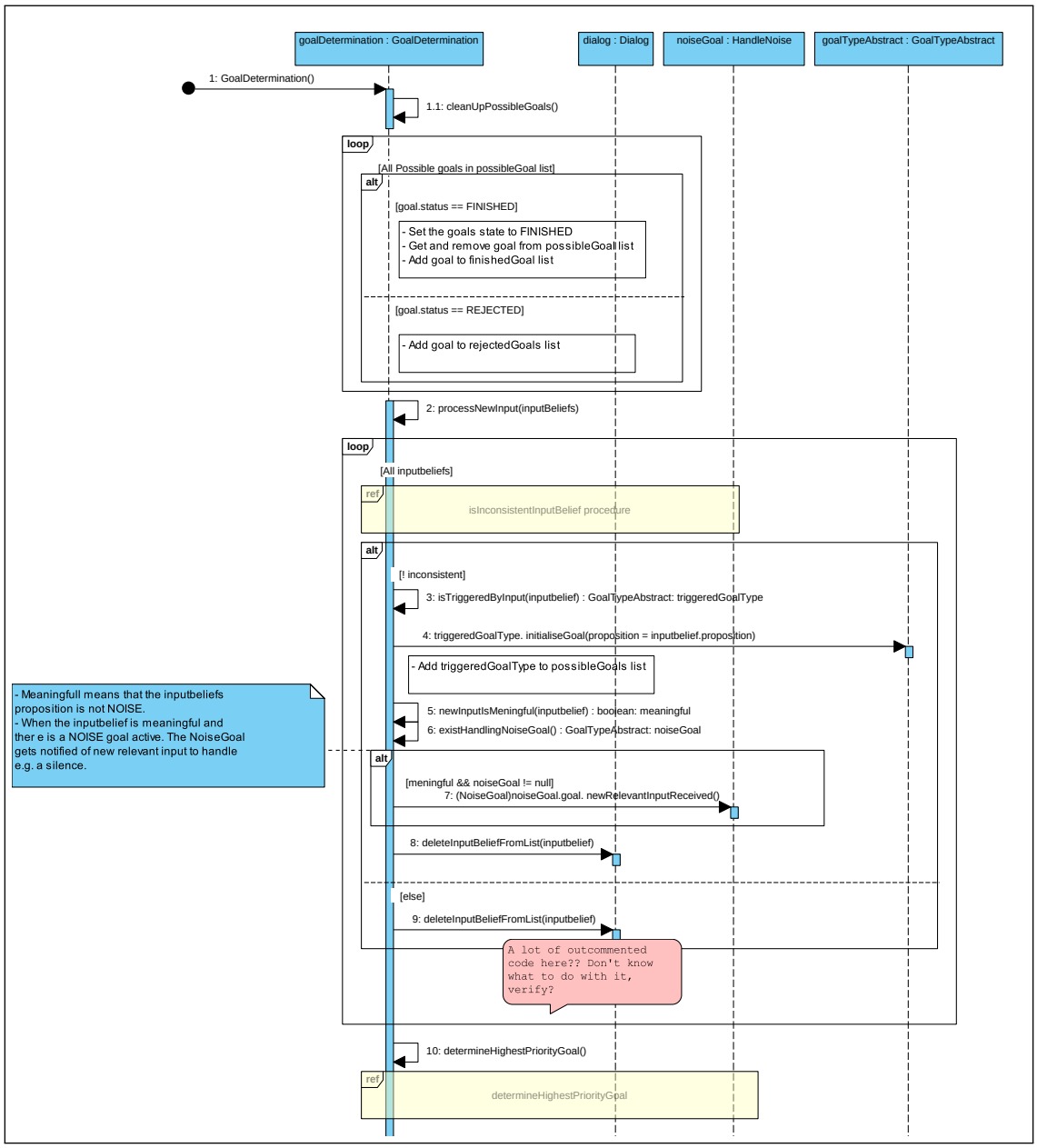
trigger Een trigger is een bepaald object wat iets activeert, in deze situatie een subdialoog. 18, 25

Utterence Een *utterence* is een herkende uitspraak. 18

XSD 'XML Schema Definitietaal' is een W3C standaard waarmee de opbouw van documenten kan worden gespecificeerd en gevalideerd. 37

A Sequence diagrammen: Abstractie dialoogverwerking




goalDetermination : GoalDetermination
dialog : Dialog
noiseGoal : HandleNoise
goalTypeAbstract : GoalTypeAbstract
1: GoalDetermination()
1.1: cleanUpPossibleGoals()
loop
[All Possible goals in possibleGoal list]
alt
[goal.status == FINISHED]
- Set the goals state to FINISHED
- Get and remove goal from possibleGoal list
- Add goal to finishedGoal list
[goal.status == REJECTED]
- Add goal to rejectedGoals list
2: processNewInput(inputBeliefs)
loop
[All inputbeliefs]
ref
isInconsistentInputBelief procedure
alt
[! inconsistent]
3: isTriggeredByInput(inputbelief) : GoalTypeAbstract: triggeredGoalType
4: triggeredGoalType. initialiseGoal(proposition = inputbelief.proposition)
- Add triggeredGoalType to possibleGoals list
5: newInputIsMeningful(inputbelief) : boolean: meaningful
6: existHandlingNoiseGoal() : GoalTypeAbstract: noiseGoal
- Meaningfull means that the inputbeliefs proposition is not NOISE.
- When the inputbelief is meaningful and ther e is a NOISE goal active. The NoiseGoal gets notified of new relevant input to handle e.g. a silence.
alt
[meningful && noiseGoal != null]
7: (NoiseGoal)noiseGoal.goal. newRelevantInputReceived()
8: deleteInputBeliefFromList(inputbelief)
[else]
9: deleteInputBeliefFromList(inputbelief)
A lot of outcommented code here?? Don't know what to do with it, verify?
10: determineHighestPriorityGoal()
ref
determineHighestPriorityGoal

B Scenario analyse tabellen

Table 17: Lost Wingman conversatie analyse

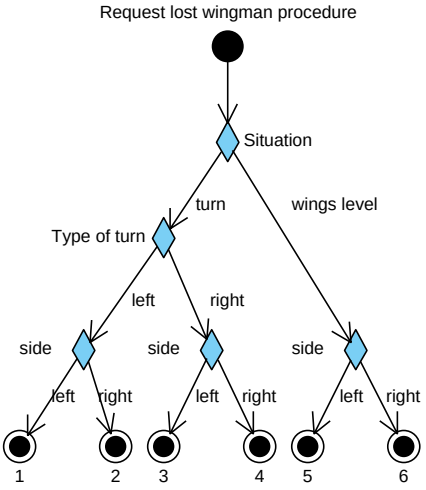
	Conversation	**Activity diagram**
1	1 U: Tell me the procedure pls?	
2	2 A: Turn or wings level? 3 U: A turn/wings level 4 A: What kind of turn/what side? 5 U: Right/left turn/side 6 A: Flying on what side? 7 U: Left/Right side	
3	8 A: inside/outside turn? or left/right side? 9 U: correct/incorrect 10 A: Incorrect not implemented. 11 U: Correct	
4	12 A: I immediately safely sep.. 13 A: I maintain the safe... 14 A: You arrange a separate.. 15 A: I transition smoothly.. 16 A: I reduce power to...etc.. 17 A: And that's it.	
5	18 A: Is this correct? 19 U: Yes it is.	

Table 18: Apparatenhulp conversatie analyse

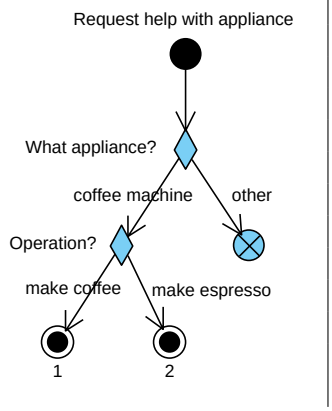
	Conversation	**Activity diagram**
1	1 U: Ashley help pls?	
2	2 A: What appliance? 3 U: Coffeemachine 4 A: What operation? 5 U: Make coffee	
4	6 A: Turn the appliance on 7 A: put cup under nozzle 8 A: Check water 9 A: Insert capsule 10 A: Press small/big cup button 11 A: Enjoy! 12 A: And that's it.	
5	13 A: Is this correct? 14 U: Yes.	

Table 19: Diagnose conversatie analyse

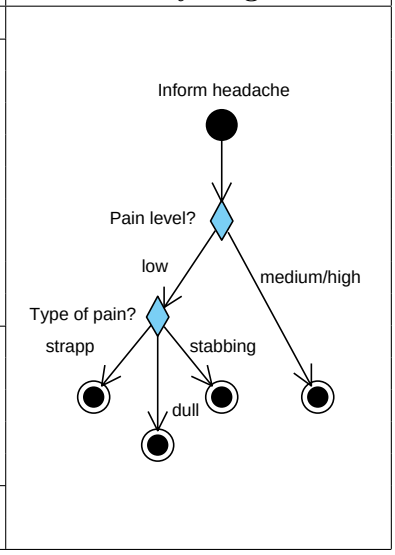
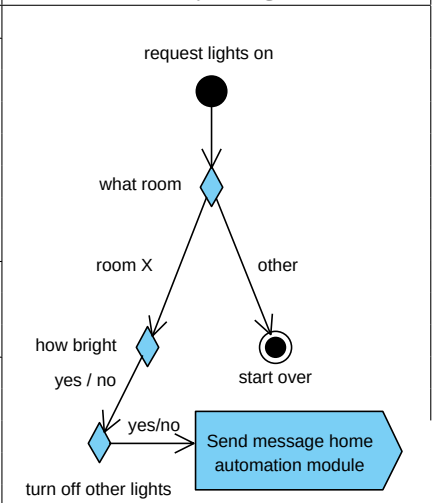
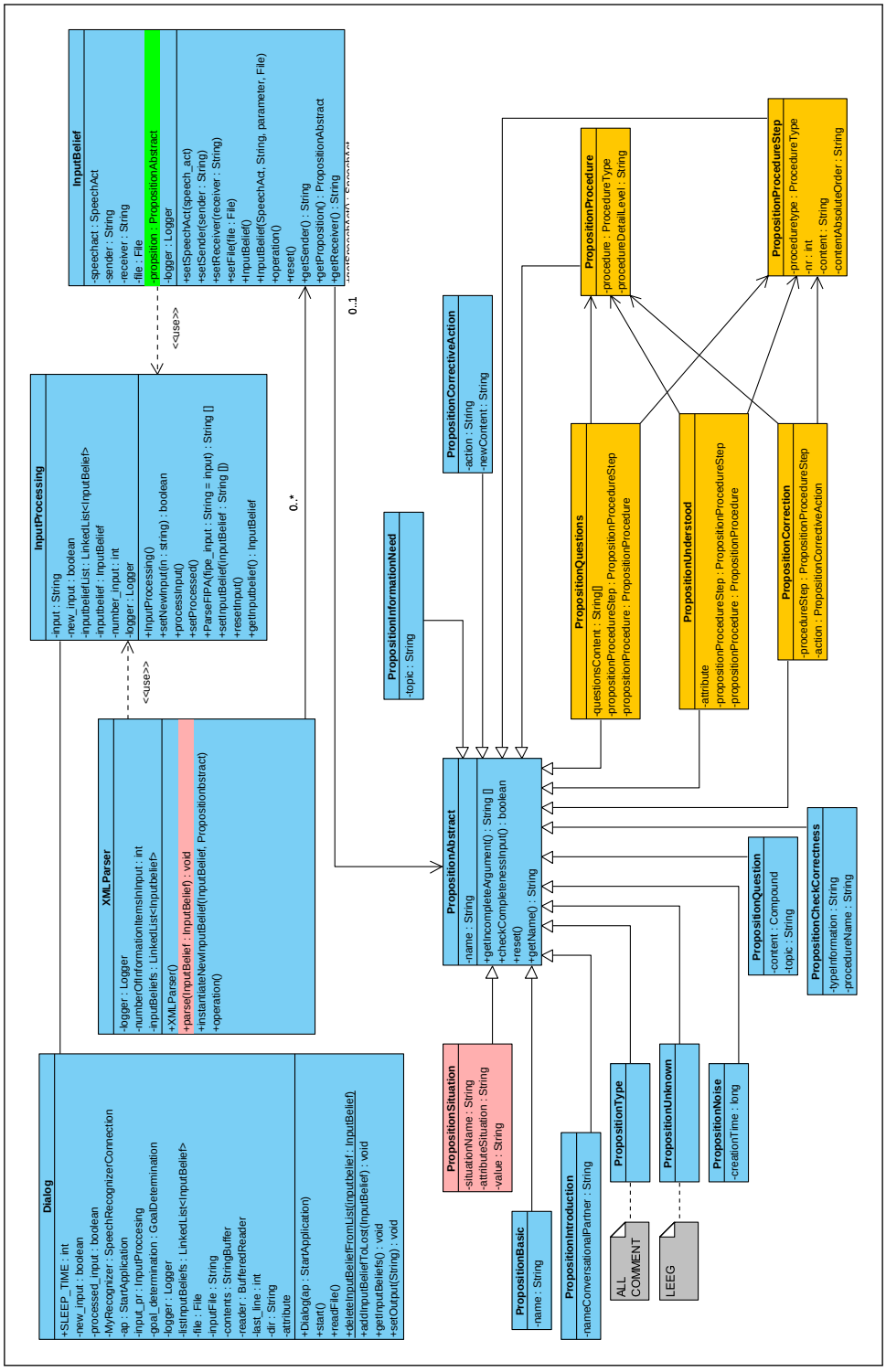
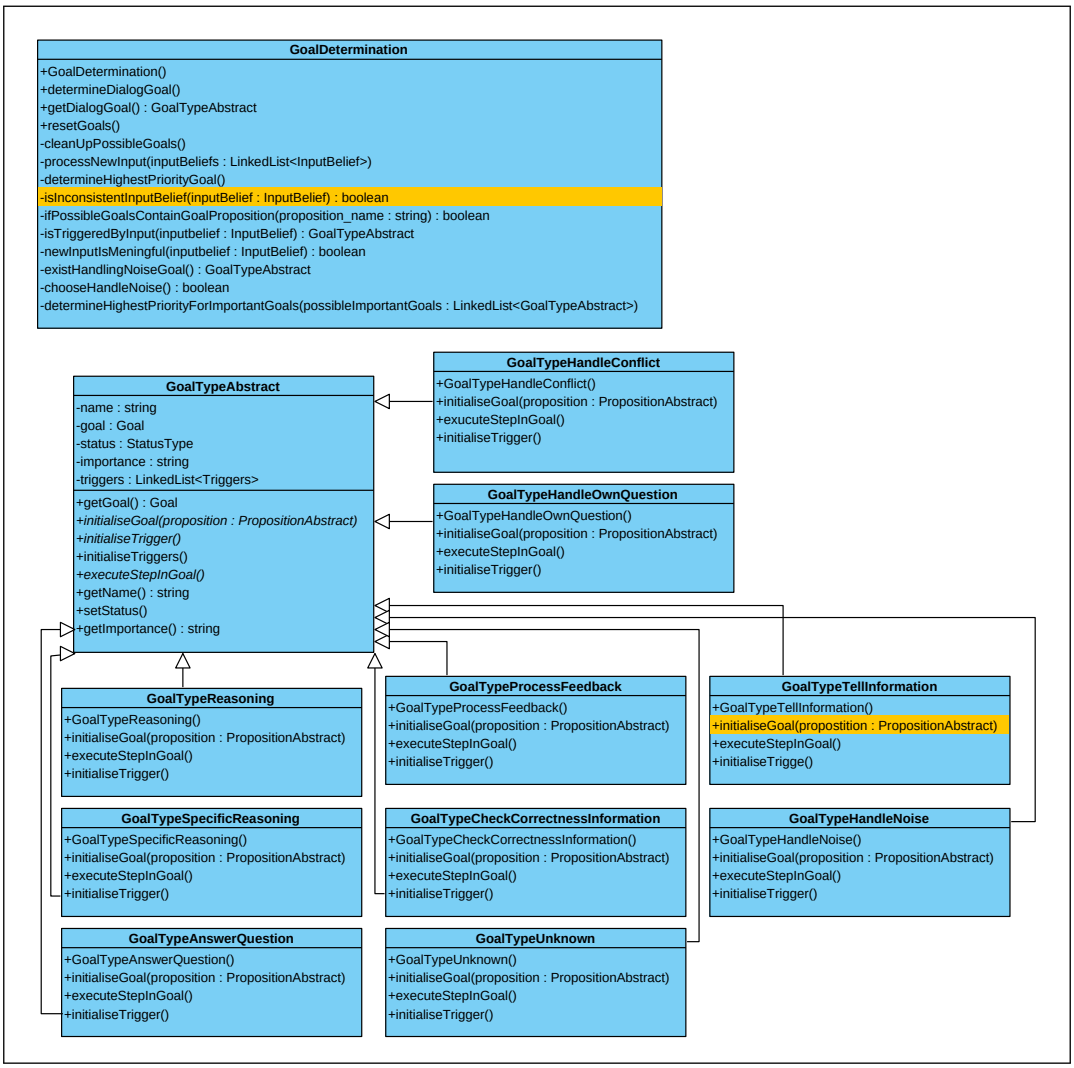
	Conversation	**Activity diagram**
1	1 U: Ashley i have a headache	
2	2 A: Pain level low/medium/high? 3 U: Low/Medium/High 4 A: High or medium END 5 U: Low 6 A: Strapp arround head/ Dull pain whole head/ severe stabbing? 7 A: Strapp/dull/stabbing 8 A: Okey	
4	9 A: Ive diagnosed migraine/ tension- or cluster headache. could be meds related 10 A: Take some rest and a asprin 11 A: you will be fine!	
5	12 A: Anything else? . 13 U: No thank you.	

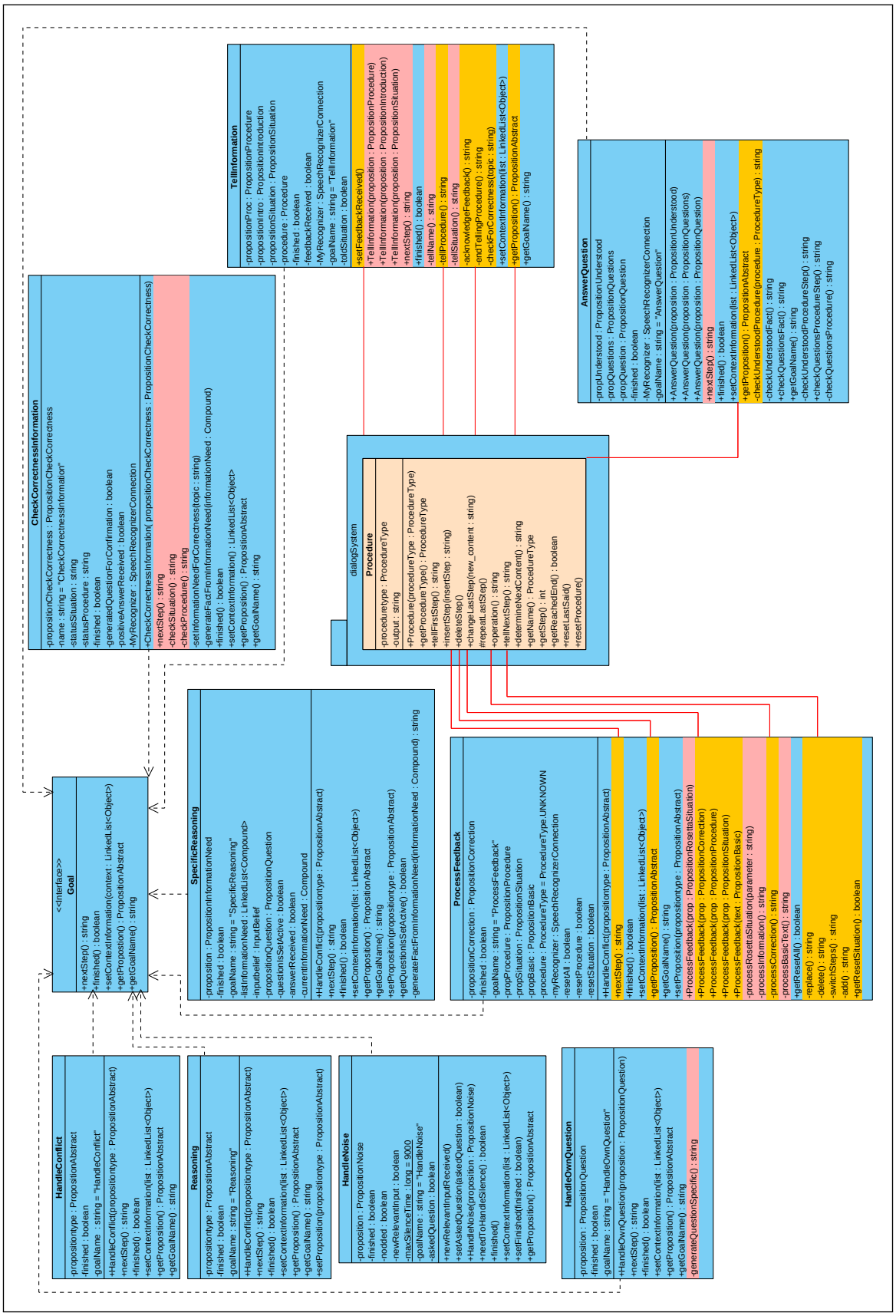
Table 20: Home automation conversatie analyse

	Conversation	**Activity diagram**
1	1 U: Ashley, lights on pls?	
2	2 A: Which room? 3 U: Livingroom/kitchen/bathroom etc. 4 A: How bright? 5 U: Not so bright. 6 A: Turn the lights in other room of 7 A: Yes pls 8 A: Okey	
4	9 A: Message light on¿home aut module 10 A: Message light off¿home aut module 11 A: Done.	
5	12 A: Anything else?. 13 U: No thank you.	

C Class Diagram: Onderdelen met specifieke dialoogverwerking




GoalDetermination
+GoalDetermination()
+determineDialogGoal()
+getDialogGoal() : GoalTypeAbstract
+resetGoals()
-cleanUpPossibleGoals()
-processNewInput(inputBeliefs : LinkedList<InputBelief>)
-determineHighestPriorityGoal()
-isInconsistentInputBelief(inputBelief : InputBelief) : boolean
-ifPossibleGoalsContainGoalProposition(proposition_name : string) : boolean
-isTriggeredByInput(inputbelief : InputBelief) : GoalTypeAbstract
-newInputIsMeningful(inputbelief : InputBelief) : boolean
-existHandlingNoiseGoal() : GoalTypeAbstract
-chooseHandleNoise() : boolean
-determineHighestPriorityForImportantGoals(possibleImportantGoals : LinkedList<GoalTypeAbstract>)
GoalTypeAbstract
-name : string
-goal : Goal
-status : StatusType
-importance : string
-triggers : LinkedList<Triggers>
+getGoal() : Goal
+initialiseGoal(proposition : PropositionAbstract)
+initialiseTrigger()
+initialiseTriggers()
+executeStepInGoal()
+getName() : string
+setStatus()
+getImportance() : string
GoalTypeHandleConflict
+GoalTypeHandleConflict()
+initialiseGoal(proposition : PropositionAbstract)
+exucuteStepInGoal()
+initialiseTrigger()
GoalTypeHandleOwnQuestion
+GoalTypeHandleOwnQuestion()
+initialiseGoal(proposition : PropositionAbstract)
+executeStepInGoal()
+initialiseTrigger()
GoalTypeReasoning
+GoalTypeReasoning()
+initialiseGoal(proposition : PropositionAbstract)
+executeStepInGoal()
+initialiseTrigger()
GoalTypeProcessFeedback
+GoalTypeProcessFeedback()
+initialiseGoal(proposition : PropositionAbstract)
+executeStepInGoal()
+initialiseTrigger()
GoalTypeTellInformation
+GoalTypeTellInformation()
+initialiseGoal(propostition : PropositionAbstract)
+executeStepInGoal()
+initialiseTrigge()
GoalTypeSpecificReasoning
+GoalTypeSpecificReasoning()
+initialiseGoal(proposition : PropositionAbstract)
+executeStepInGoal()
+initialiseTrigger()
GoalTypeCheckCorrectnessInformation
+GoalTypeCheckCorrectnessInformation()
+initialiseGoal(proposition : PropositionAbstract)
+executeStepInGoal()
+initialiseTrigger()
GoalTypeHandleNoise
+GoalTypeHandleNoise()
+initialiseGoal(proposition : PropositionAbstract)
+executeStepInGoal()
+initialiseTrigger()
GoalTypeAnswerQuestion
+GoalTypeAnswerQuestion()
+initialiseGoal(proposition : PropositionAbstract)
+executeStepInGoal()
+initialiseTrigger()
GoalTypeUnknown
+GoalTypeUnknown()
+initialiseGoal(proposition : PropositionAbstract)
+executeStepInGoal()
+initialiseTrigger()


CheckCorrectnessInformation
-propositionCheckCorrectness : PropositionCheckCorrectness
-name : string = "CheckCorrectnessInformation"
-statusSituation : string
-statusProcedure : string
-finished : boolean
-generatedQuestionForConfirmation : boolean
-positiveAnswerReceived : boolean
-MyRecognizer : SpeechRecognizerConnection
+CheckCorrectnessInformation(propositionCheckCorrectness : PropositionCheckCorrectness)
+nextStep() : string
-checkSituation() : string
-checkProcedure() : string
-setInformationNeedForCorrectness(topic : string)
-generateFactFromInformationNeed(informationNeed : Compound)
+finished() : boolean
+setContextInformation() : LinkedList<Object>
+getProposition() : PropositionAbstract
+getGoalName() : string
TellInformation
-propositionProc : PropositionProcedure
-propositionIntro : PropositionIntroduction
-propositionSituation : PropositionSituation
-procedure : Procedure
-finished : boolean
-feedbackReceived : boolean
-MyRecognizer : SpeechRecognizerConnection
-goalName : string = "TellInformation"
-toldSituation : boolean
+setFeedbackReceived()
+TellInformation(proposition : PropositionProcedure)
+TellInformation(proposition : PropositionIntroduction)
+TellInformation(proposition : PropositionSituation)
+nextStep() : string
+finished() : boolean
-tellName() : string
-tellProcedure() : string
-tellSituation() : string
-acknowledgeFeedback() : string
-endTellingProcedure() : string
-checkForCorrectness(topic : string)
+setContextInformation(list : LinkedList<Object>)
+getProposition() : PropositionAbstract
+getGoalName() : string
AnswerQuestion
-propUnderstood : PropositionUnderstood
-propQuestions : PropositionQuestions
-propQuestion : PropositionQuestion
-finished : boolean
-MyRecognizer : SpeechRecognizerConnection
-goalName : string = "AnswerQuestion"
+AnswerQuestion(proposition : PropositionUnderstood)
+AnswerQuestion(proposition : PropositionQuestions)
+AnswerQuestion(proposition : PropositionQuestion)
+nextStep() : string
+finished() : boolean
+setContextInformation(list : LinkedList<Object>)
+getProposition() : PropositionAbstract
-checkUnderstoodProcedure(procedure : ProcedureType) : string
-checkUnderstoodFact() : string
+checkQuestionsFact() : string
+getGoalName() : string
-checkUnderstoodProcedureStep() : string
+checkQuestionsProcedureStep() : string
-checkQuestionsProcedure() : string
dialogSystem
Procedure
-proceduretype : ProcedureType
-output : string
+Procedure(proceduretype : ProcedureType)
+getProcedureType() : ProcedureType
+tellFirstStep() : string
+insertStep(insertStep : string)
+deleteStep()
+changeLastStep(new_content : string)
#repeatLastStep()
+operation() : string
+tellNextStep() : string
+determineNextContent() : string
+getName() : ProcedureType
+getStep() : int
+getReachedEnd() : boolean
+resetLastSaid()
+resetProcedure()
<<Interface>>
Goal
+nextStep() : string
+finished() : boolean
+setContextInformation(context : LinkedList<Object>)
+getProposition() : PropositionAbstract
+getGoalName() : string
SpecificReasoning
-proposition : PropositionInformationNeed
-finished : boolean
-goalName : string = "SpecificReasoning"
-listInformationNeed : LinkedList<Compound>
-inputbelief : InputBelief
-propositionQuestion : PropositionQuestion
-questionIsSetoActive : boolean
-answerReceived : boolean
-currentInformationNeed : Compound
+HandleConflict(propositiontype : PropositionAbstract)
+nextStep() : string
+finished() : boolean
+setContextInformation(list : LinkedList<Object>)
+getProposition() : PropositionAbstract
+getGoalName() : string
+setQuestionIsSetActive() : boolean
-generateFactFromInformationNeed(informationNeed : Compound) : string
ProcessFeedback
-propositionCorrection : PropositionCorrection
-finished : boolean
-goalName : string = "ProcessFeedback"
-propProcedure : PropositionProcedure
-propSituation : PropositionSituation
-propBasic : PropositionBasic
-procedure : ProcedureType = ProcedureType.UNKNOWN
-myRecognizer : SpeechRecognizerConnection
-resetAll : boolean
-resetProcedure : boolean
-resetSituation : boolean
+HandleConflict(propositiontype : PropositionAbstract)
+nextStep() : string
+finished() : boolean
+setContextInformation(list : LinkedList<Object>)
+getProposition() : PropositionAbstract
+getGoalName() : string
+setProposition(propositiontype : PropositionAbstract)
+ProcessFeedback(prop : PropositionRosettaSituation)
+ProcessFeedback(prop : PropositionCorrection)
+ProcessFeedback(prop : PropositionProcedure)
+ProcessFeedback(prop : PropositionSituation)
+ProcessFeedback(text : PropositionBasic)
-processRosettaSituation(parameter : string)
-processInformation() : string
-processCorrection() : string
-processBasic(text) : string
+getResetAll() : boolean
-replace() : string
-delete() : string
-switchSteps() : string
-add() : string
+getResetSituation() : boolean
HandleConflict
-propositiontype : PropositionAbstract
-finished : boolean
-goalName : string = "HandleConflict"
+HandleConflict(propositiontype : PropositionAbstract)
+nextStep() : string
+finished() : boolean
+setContextInformation(list : LinkedList<Object>)
+getProposition() : PropositionAbstract
+getGoalName() : string
+setProposition(propositiontype : PropositionAbstract)
Reasoning
-propositiontype : PropositionAbstract
-finished : boolean
-goalName : string = "Reasoning"
+HandleConflict(propositiontype : PropositionAbstract)
+nextStep() : string
+finished() : boolean
+setContextInformation(list : LinkedList<Object>)
+getProposition() : PropositionAbstract
+getGoalName() : string
+setProposition(propositiontype : PropositionAbstract)
HandleNoise
-proposition : PropositionNoise
-finished : boolean
-nodded : boolean
-newRelevantInput : boolean
-maxSilenceTime : long = 9000
-goalName : string = "HandleNoise"
-askedQuestion : boolean
+newRelevantInputReceived()
+setAskedQuestion(askedQuestion : boolean)
+HandleNoise(proposition : PropositionNoise)
+resetToHandleSilence() : boolean
+finished()
+setContextInformation(list : LinkedList<Object>)
+setFinished(finished : boolean)
+getProposition() : PropositionAbstract
HandleOwnQuestion
-proposition : PropositionQuestion
-finished : boolean
-goalName : string = "HandleOwnQuestion"
+HandleOwnQuestion(proposition : PropositionQuestion)
+nextStep() : string
+finished() : boolean
+setContextInformation(list : LinkedList<Object>)
+getProposition() : PropositionAbstract
+getGoalName() : string
-generateQuestionSpecific() : string

D Mogelijk verloop hoofdpijn diagnose scenario

User	**Reasoning**	**Ashley**
Ashley I have a headache!	Received input belief: Inform subdialog diagnose_headache	
	Add_belief: current_subdialog = diagnose_headache	
	Check information needs = attack_duration	
	Determine corresponding question	Do the attacks last more than 4 hours or less?
I guess they last more than 4 hours	Process the Feedback / Add belief: Attack_duration = more_then_4_hours / Update information needs	
	Check information needs = Type of turn	
	Determine corresponding question	Can you describe your pain level in term of high, medium or low?
I think it is a medium pain	Process the Feedback / Add belief: level = medium / Update information needs	
	Check information needs = symptom	
	Determine corresponding question	Did you have any of the symptoms nauseousness, sensitive to light and sound or did throw up?
I am feeling nauseouss right now	Process the Feedback / Add belief: symptom = nauseous / Update information needs	
	Check information needs = none. situation = migraine	
	Execute intention (belongs to situation migraine)	
	Determine step in intention	
	Get corresponding sentence	This is step N of the intention
	End of executing intention reached	
	add correctness information need	
	Check information needs = correctness	
	Determine corresponding question	Can i help you with anything else?
Yes can you turn off the lights?	Process the Feedback / Add belief: correctness = correct / Update information needs	
	Determine messages to send	trigger home automation subdialog.
	Determine acknowledge sentence	Okay!

E XML hoofdpijndiagnose

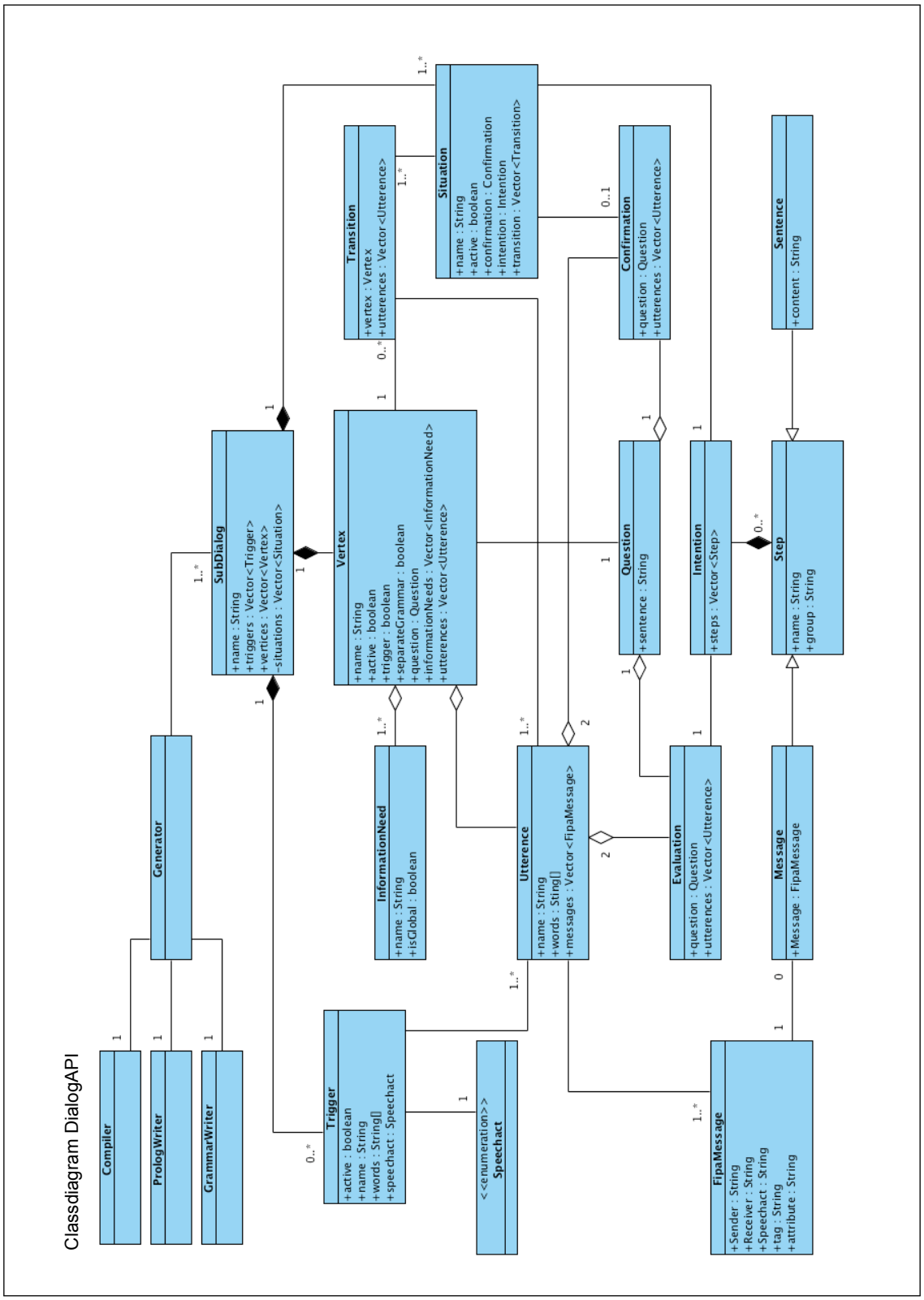
```
<Dialogsystem>
    <Subdialog Name="headache_diagnose">
        <Triggers>
            <Trigger Active="True" Name="defaulttrigger" Receiver="Ashley" Sender="User"
                Speechact="Inform">
                <Words>i have a headache | my head is beating</Words>
            </Trigger>
        </Triggers>
        <SpecifySituation>
            <Vertices>
                <Vertex Active="True" InformationNeed="headache_duration"
                    InformationNeedIsGlobal="true" Name="V0" SeparateGrammar="true" Trigger="false">
                    <Body>
                        <QuestionSentence>Last the headace less or more than 4
                            hours?</QuestionSentence>
                        <Utterences>
                            <Utterence Name="more_than_4_hours">
                                <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                                    <Content>
                                        <Tag>situation</Tag>
                                        <Tag>headache_duration</Tag>
                                        <Attribute>more_than_4_hours</Attribute>
                                    </Content>
                                </Message>
                                <Words>more than 4 hours | definitely more</Words>
                                <Transition>V1</Transition>
                            </Utterence>
                            <Utterence Name="less_than_4_hours">
                                <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                                    <Content>
                                        <Tag>situation</Tag>
                                        <Tag>headache_duration</Tag>
                                        <Attribute>less_than_4_hours</Attribute>
                                    </Content>
                                </Message>
                                <Words>less than 4 hours | no less</Words>
                                <Transition>V4</Transition>
                            </Utterence>
                        </Utterences>
                    </Body>
                </Vertex>
                <Vertex Active="True" InformationNeed="headache_pain_level"
                    InformationNeedIsGlobal="false" Name="V1" SeparateGrammar="true" Trigger="false">
                    <Body>
                        <QuestionSentence>Describe your pain level in terms of high, medium or
                            low?</QuestionSentence>
                        <Utterences>
                            <Utterence Name="pain_level_high">
                                <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                                    <Content>
                                        <Tag>situation</Tag>
                                        <Tag>pain_level</Tag>
                                        <Attribute>high</Attribute>
                                    </Content>
                                </Message>
                                <Words>high</Words>
                                <Transition>V3</Transition>
                            </Utterence>
                            <Utterence Name="pain_level_medium">
                                <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                                    <Content>
                                        <Tag>situation</Tag>
                                        <Tag>pain_level</Tag>
                                        <Attribute>medium</Attribute>
                                    </Content>
                                </Message>
                                <Words>medium</Words>
```

```
                    <Transition>V3</Transition>
                </Utterence>
                <Utterence Name="pain_level_low">
                    <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                        <Content>
                            <Tag>situation</Tag>
                            <Tag>pain_level</Tag>
                            <Attribute>low</Attribute>
                        </Content>
                    </Message>
                    <Words>low</Words>
                    <Transition>V2</Transition>
                </Utterence>
            </Utterences>
        </Body>
    </Vertex>
    <Vertex Active="True" InformationNeed="headache_symptom"
        InformationNeedIsGlobal="false" Name="V2" SeparateGrammar="true" Trigger="false">
        <Body>
            <QuestionSentence>Does the pain increase during physical
                activity?</QuestionSentence>
            <Utterences>
                <Utterence Name="increase_activity">
                    <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                        <Content>
                            <Tag>situation</Tag>
                            <Tag>symptom</Tag>
                            <Attribute>increase_activity</Attribute>
                        </Content>
                    </Message>
                    <Words>yes | indeed | increase activity</Words>
                    <Transition>V3</Transition>
                </Utterence>
                <Utterence Name="not_increase_activity">
                    <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                        <Content>
                            <Tag>situation</Tag>
                            <Tag>symptom</Tag>
                            <Attribute>not_increase_activity</Attribute>
                        </Content>
                    </Message>
                    <Words>no | not</Words>
                    <Transition>none</Transition>
                </Utterence>
            </Utterences>
        </Body>
    </Vertex>
    <Vertex Active="True" InformationNeed="headache_symptom"
        InformationNeedIsGlobal="false" Name="V3" SeparateGrammar="true" Trigger="false">
        <Body>
            <QuestionSentence>Did you have any of the symptoms nauseousness, sensitive
                to light and sound or did throw up?</QuestionSentence>
            <Utterences>
                <Utterence Name="nauseous">
                    <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                        <Content>
                            <Tag>situation</Tag>
                            <Tag>symptom</Tag>
                            <Attribute>nauseous</Attribute>
                        </Content>
                    </Message>
                    <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                        <Content>
                            <Tag>situation</Tag>
                            <Tag>symptom</Tag>
                            <Attribute>throw_up</Attribute>
                        </Content>
```

```
                        </Message>
                        <Words>nauseous | throw up</Words>
                        <Transition>migraine</Transition>
                    </Utterence>
                    <Utterence Name="sensitive">
                        <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                            <Content>
                                <Tag>situation</Tag>
                                <Tag>symptom</Tag>
                                <Attribute>sensitive</Attribute>
                            </Content>
                        </Message>
                        <Words>sensitive</Words>
                        <Transition>migraine</Transition>
                    </Utterence>
                    <Utterence Name="not_nauseous">
                        <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                            <Content>
                                <Tag>situation</Tag>
                                <Tag>symptom</Tag>
                                <Attribute>not_nauseous</Attribute>
                            </Content>
                        </Message>
                        <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                            <Content>
                                <Tag>situation</Tag>
                                <Tag>symptom</Tag>
                                <Attribute>not_sensitive</Attribute>
                            </Content>
                        </Message>
                        <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                            <Content>
                                <Tag>situation</Tag>
                                <Tag>symptom</Tag>
                                <Attribute>not_throw_up</Attribute>
                            </Content>
                        </Message>
                        <Words>neither</Words>
                        <Transition>posibly_migraine</Transition>
                    </Utterence>
                </Utterences>
            </Body>
        </Vertex>
    </Vertices>
    <Situations>
        <Situation Active="True" Name="migraine"/>
        <Situation Active="True" Name="posibly_migraine"/>
    </Situations>
</SpecifySituation>
<ConfirmSituation>
    <Confirmation Active="True" Situation="migraine">
        <Question>Is it correct that your pain level is medium or high and that you are
            feeling nauseous?</Question>
        <AnswerCorrect>
            <Words>Yes | correct</Words>
            <Message Receiver="Ashley" Sender="User" Speechact="Inform">
                <Content>
                    <Tag>feedback</Tag>
                    <Attribute>correct</Attribute>
                </Content>
            </Message>
        </AnswerCorrect>
        <AnswerIncorrect>
            <Words>no</Words>
            <Message Receiver="Ashley" Sender="User" Speechact="Inform">
                <Content>
                    <Tag>feedback</Tag>
```

```
<Dialogsystem>
    <Subdialog Name="headache_diagnose">
        <Triggers>
            <Trigger Active="True" Name="defaulttrigger" Receiver="Ashley" Sender="User"
                Speechact="Inform">
                <Words>i have a headache | my head is beating</Words>
            </Trigger>
        </Triggers>
        <SpecifySituation>
            <Vertices>
                <Vertex Active="True" InformationNeed="headache_duration"
                    InformationNeedIsGlobal="true" Name="V0" SeparateGrammar="true" Trigger="false">
                    <Body>
                        <QuestionSentence>Last the headace less or more than 4
                            hours?</QuestionSentence>
                        <Utterences>
                            <Utterence Name="more_than_4_hours">
                                <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                                    <Content>
                                        <Tag>situation</Tag>
                                        <Tag>headache_duration</Tag>
                                        <Attribute>more_than_4_hours</Attribute>
                                    </Content>
                                </Message>
                                <Words>more than 4 hours | definitely more</Words>
                                <Transition>V1</Transition>
                            </Utterence>
                            <Utterence Name="less_than_4_hours">
                                <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                                    <Content>
                                        <Tag>situation</Tag>
                                        <Tag>headache_duration</Tag>
                                        <Attribute>less_than_4_hours</Attribute>
                                    </Content>
                                </Message>
                                <Words>less than 4 hours | no less</Words>
                                <Transition>V4</Transition>
                            </Utterence>
                        </Utterences>
                    </Body>
                </Vertex>
                <Vertex Active="True" InformationNeed="headache_pain_level"
                    InformationNeedIsGlobal="false" Name="V1" SeparateGrammar="true" Trigger="false">
                    <Body>
                        <QuestionSentence>Describe your pain level in terms of high, medium or
                            low?</QuestionSentence>
                        <Utterences>
                            <Utterence Name="pain_level_high">
                                <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                                    <Content>
                                        <Tag>situation</Tag>
                                        <Tag>pain_level</Tag>
                                        <Attribute>high</Attribute>
                                    </Content>
                                </Message>
                                <Words>high</Words>
                                <Transition>V3</Transition>
                            </Utterence>
                            <Utterence Name="pain_level_medium">
                                <Message Receiver="User" Sender="Ashley" Speechact="Inform">
                                    <Content>
                                        <Tag>situation</Tag>
                                        <Tag>pain_level</Tag>
                                        <Attribute>medium</Attribute>
                                    </Content>
                                </Message>
                                <Words>medium</Words>
```

F Klassendiagram dialoog generator



F.1 Omschrijving bij het klassendiagram

De dialoog generator zorgt ervoor dat de Prolog constructies worden gegenereerd in de logische databases en dat de grammar file structuur wordt aangemaakt. Daarnaast voorziet de generator in tools voor het compileren van het systeem en het structureren van de benodigde bestanden.

DialogGenerator
De klasse die voorziet in het genereren van een dialoogsysteem met de ontwikkelde dialoog. De klasse heeft als verantwoordelijkheden: het verwerken van de XML file, het aanmaken van de benodigde objecten. De klasse heeft één of meer referenties naar de klasse Subdialoog, hij kent een Compiler, een PrologWriter en een GrammarWriter. De klasse voorziet in methodes om met deze combinatie een dialoogsysteem te genereren.

Compiler
Wanneer het systeem volledig is ingeladen voorziet de Compiler klasse in het compileren van het dialoogsysteem, daarnaast zorg de klasse ervoor dat alle benodigde bestanden worden gestructureerd in een archief bestand.

PrologWriter
De *Prolog* bestanden zijn gestructureerd opgebouwd, er zijn tags te vinden waarbinnen de code elementen gegenereerd kunnen worden. De DialogGenerator zal de instanties van Subdialog meegegeven als parameter, deze worden dan verwerkt in de *Prolog* bestanden. Daarnaast heeft deze een controlerende functie, die er voor zorgt dat de *Prolog* data op de juiste manier wordt gegenereerd.

GrammarWriter
Wanneer de DialogGenerator deze klasse aanroept worden de nodige *grammar* bestanden aangemaakt. Op de eerste plaats zal deze klasse het selector bestand genereren, waarin de *triggers* gespecificeerd staan voor de subdialogen welke door de spraakherkenner *getriggerd* kunnen worden. De volgende stap is het genereren van bestanden voor de onafhankelijke subdialogen. Een subdialoog kan alle *grammar* in één bestand hebben gegenereerd of er kan voor elke Vertex een eigen bestand worden aangemaakt. Als laatste wordt er per subdialoog een bestand voor stap confirm situation en evaluate intention aangemaakt.

Subdialoog
De klasse Subdialoog is een overkoepelende klasse van alle data behoren tot een subdialoog. Een subdialoog heeft een naam en één of meer referenties naar instanties van de klasse Vertex. Een subdialoog kan intern of extern *getriggerd* worden, in het tweede geval zijn er referenties nodig naar instanties van de klasse Trigger. Een subdialoog kan meerdere *triggers* hebben.

Trigger
Subdialogen kunnen worden geïnitieerd door een *trigger*, een Trigger object bevat een *utterence* en een *speechact*. De verantwoordelijkheid van deze klasse is het aanmaken van een FIPAmessage object en deze toe te voegen aan het Utterence object.

Vertex
Een Vertex is een klasse wat een knooppunt in de subdialoog representeert, het staat voor een keuze die gemaakt wordt en de gevolgen die daar bij horen. Op basis van een *utterence* en één van de mogelijk voorgaande transities, wordt de volgende transitie bepaald. Daarnaast bevat de Vertex klasse een InformationNeed object, de *information need* gecombineerd met de *utterence* vormen een feit voor de *Prolog* database. Behorend tot de Vertex klasse is een instantie van de klasse Question. Een ander belangrijk attribuut van deze klasse is; boolean generateGrammar, wanneer deze true is zal er een appart *grammar* bestand worden gegenereerd voor deze vertex.

Utterence
In de *grammar* bestanden worden woord combinaties aangemaakt, een *utterence* is een label voor mogelijke woordcombinatie waar het systeem op reageert. De Utterence klasse bevat een string met woordcombinaties en een instantie van een FIPAmessage.

InformationNeed
De klasse die een informatie behoefte representeert behorende tot een vertex.

Transition
De klasse Transition heeft één referentie naar een Vertex instantie en de daarbij horende Utterence instanties die invulling geven aan de *information needs* van de vertex.

FIPAmessage
Een klasse wat een *FIPA ACL* bericht typeert.

Speechact
Een enumeratie van de *speechacts* die voorkomen in het systeem.

Situation
De klasse met de specificatie van de eind-situaties. Hier staan de situaties benoemd, de transitie hoe deze situatie wordt gekozen en de relaties naar de Confimation en de Intention die bij deze situatie horen.

Confirmation
De klasse voor de bevestiging van de situatie. Er is een object Question met de vraagstelling en een tweetal antwoorden (klasse Utterence).

Evaluation
De klasse voor de bevestiging van de uitvoer van de intentie. Er is een object Question met de vraagstelling en een tweetal antwoorden (klasse Utterence).

Intention
De klasse intentie bevat een lijst met objecten van het type Step.

Step
De Step klasse is een superklasse van de klasse Sentence en Message. Het bevat onder andere een attribuut naam en een attribuut group.

Sentence
Een klasse dat een output zin typeert.

Question
Een klasse dat een output vraag typeert.

Message
Een klasse dat gebruikt wordt voor interne en externe berichtgeving. De klasse heeft daarom één instantie van de klasse FIPAmessage.

References

[1] M. Wooldridge and N. R. Jenning. *Intelligent agents: theory and practice.* Knowledge Eng. Rev., vol. 10(2), pp. 115152, 1995

[2] Bratman, M. E. *Intention, Plans, and Practical Reason.* CSLI Publications, (1999) [1987], ISBN 1-57586-192-5.

[3] A. S. Rao and M. P. Georgeff. *Modeling Rational Agents within a BDI-Architecture.* In Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning, pages 473-484, 1991

[4] James R. Glasss. *Challanges for sproken dialogue systems*Spoken Language Systems Group MIT Laboratory for Computer Science, Cambridge, MA 02139. http://www.sls.lcs.mit.edu

[5] Giovanni Flammia. *Discourse Segmentation Of Spoken Dialogue:An Empirical Approach* Massachusetts Institute of Technology, pp. 27-36, 49-51, pp. 65-97, 1998

[6] Schwaber, Ken. *Agile Project Management with Scrum.* Redmond Washington: Microsoft Press, 2004, ISBN 0-7356-1993-x.

[7] Oetiker, Partl, Hyna, Schlegl. *The Not So Short Introduction to LateX 2e.* Cambridge: GNU General Public License, 1998.

[8] Endriss, Ulle. *Lecture Notes, An Introduction to Prolog Programming.* London: King's college, 2000.

[9] SWI-Prolog's home(*SWI*). swi-prolog.org, 3 May 2011. http://www.swi-prolog.org

[10] Van Doesburg, Willem(*Ashley, a virtuele personal assistent*). tno.nl, version October 2009. May 25, 2011. http://www.tno.nl/content.cfm?context=overtno&content=nieuwsbericht&laag1=37&laag2=69&item_id=2009-10-29%2011:49:09.0.

[11] Nauts, Pim. *Taking Turns in Flying With a Virtual Wingman. Human-machine interaction for military aviation.* Tilburg: Univirsity Department of Communication and Information Sciences. July 2010.