FATT
FAiry Tale Teller

Dirkjan Bussink, Tom Nijmeijer, Matthijs Ooms

2 september 2005

Abstract

Dit verslag biedt een inzicht in het door ons ontwikkelde systeem genaamd FATT. FATT staat voor FAiry Tale Teller en is zoals de naam al aangeeft, een sprookjesverteller. Het nieuwe hieraan is, is dat het een systeem is dat ontwikkeld is voor het Nederlands en dat het niet alleen een verhaal verteld, maar er ook gebaren en bewegingen bij maakt zodat het geheel natuurlijk en levendig overkomt.

FATT is een systeem dat Nederlandstalige teksten verhalen kan vertellen. De gebruiker kan in een bestand een verhaaltje zetten, dat vervolgens met behulp van een agent levendig verteld wordt. Deze agent maakt tijdens het vertellen gebaren om het verhaal te ondersteunen.

De inspiratie voor FATT komt van de BEAT toolkit, een systeem om agents iets te laten vertellen. Dit is echter in het Engels, FATT daarentegen is in het Nederlands. Verder wordt er externe software gebruikt om te kunnen PoS-taggen en syntactisch te parsen. De laatstgenoemde is de Alpino parser die ontwikkeld is op de Rijksuniversiteit van Groningen. Natuurlijk hebben we ook ideeën ontleend uit de BEAT toolkit [4] en tevens uit het boek van Jurafsky en Martin [6].

Op dit moment ondersteunt FATT een aantal verschillende gebaren. Allereerst zijn er de iconic gestures die bepaalde concepten weergeven en de metaphoric gestures. Beide kunnen door middel van keyword-spotting tot op zekere hoogte geïmplementeerd worden. Dit is op dit moment mogelijk binnen FATT, ware het niet dat er nog geen grote database is met dergelijke gegevens. Verder zijn er nog de deictic gestures en de beat gestures. Hiervan zijn alleen de beat gestures geïmplementeerd, deictic gestures komen zo goed als niet voor bij het vertellen van sprookjes.

Behalve deze standaard gebaren is er ook nog ondersteuning voor de detectie van sprekers. Dit houdt in dat bij quotes van bepaalde karakters in een sprookje bepaald wordt wie dat zegt. Hiermee kunnen posture shifts van de verteller agent gerealiseerd worden en is het ook mogelijk om b.v. verschillende stemmen voor veschillende karakters te gebruiken.

Het uiteindelijk door ons ontwikkelde systeem is duidelijk een prototype dat nog veel ruimte biedt voor uitbreidingen. De eerste opzet geeft naar onze mening echter wel een goed idee voor wat er mogelijk is en hoe virtuele verhalenvertellers verbeterd kunnen worden. Het systeem zou in de huidige vorm redelijk snel aan b.v. de Virtual Storyteller gekoppeld kunnen worden. Een betere agent is voor een redelijke visualisatie echter wel een vereiste.

Inhoudsopgave

Hoofdstuk 1

Inleiding

In het vakgebied Human Media Interaction word veel onderzoek gedaan naar conversational agents. Een belangrijke subgroep van deze conversational agents zijn de embodied conversational agents. Deze ECA's hebben naast de mogelijkheid om te interacteren met een gebruiker via tekst of spraak, één of andere visualisatie. Deze visualisatie heeft meestal als doel om de interactie te verrijken. Dit heeft als gevolg dat de gebruiker de agent meer als een daadwerkelijk persoon zal zien wat de effectiviteit van de interactie verbeterd.

Een belangrijke vorm van visualisatie is het maken van gebaren. Meestal worden gebaren gegenereerd aan de hand van hogere orde representaties van een gesprek. Een andere interessante en waarschijnlijk moeilijkere optie is om de gebaren te genereren aan de hand van gewone tekst. Één van de enigste systemen die dit doet is het BEAT systeem. Het BEAT systeem onderwerpt de tekst aan een uitgebreide diagnose om onder andere te bepalen wat de theme en rheme gedeelten van een zin zijn. Aan de hand van deze diagnose worden de gebaren waaronder beats toegevoegd.

In navolging van het BEAT systeem hebben wij een systeem ontworpen en geïmplementeerd dat eenzelfde functie vervuld met één belangrijk verschil. Waar BEAT gemaakt is voor de Engelse taal is ons systeem ontworpen voor het Nederlands. Verder is ons systeem vanwege praktische redenen beperkt tot het vertellen van sprookjes. Een domein wat goed aansluit bij het Virtual Storyteller systeem dat ontwikkeld wordt op de Universiteit Twente. We hopen ons systeem uiteindelijk te kunnen koppelen aan dit systeem om zo een volledig systeem te verwezenlijken.

In dit verslag zullen we allereerst wat dieper ingaan op het BEAT systeem en de Virtual Storyteller in het hoofdstuk `Bestaande Systemen` [2]. Vervolgens zullen we het ontwerp van ons systeem bespreken in het *Design* [3] hoofdstuk. Hierbij zal aan de orde komen waarom we ons beperkt hebben tot het domein van het vertellen van sprookjes en welke gebaren we hebben geïmplementeerd. In het derde hoofdstuk zullen we de gebruikte externe technologieën [4] bespreken. Daarna zullen we in hoofdstuk [5] de daadwerkelijke implementatie bespreken. Hierin bespreken we welke problemen we zijn tegengekomen tijdens het implementeren en hoe we deze opgelost hebben. Als één na laatste bespreken we in het hoofdstuk Evaluatie [6] de problemen die we niet hebben kunnen oplossen en ons eigen oordeel van het uiteindelijke systeem. Tenslotte zullen we in het hoofdstuk Toekomstig Werk [7] aangeven hoe het systeem verbetert zou kunnen worden.

Hoofdstuk 2

Bestaande Systemen

In dit hoofdstuk zullen we een tweetal complete systemen bespreken die enig verband houden met het door ons beoogde systeem. Allereerst zullen we het hebben over BEAT. Dit systeem is de inspiratiebron van ons systeem en heeft dan ook dezelfde features als we in ons systeem zouden willen. Het tweede systeem is de Virtual Storyteller. Een systeem dat net als de onze gericht is op het domein van de sprookjes.

2.1 BEAT

BEAT [4] is een vertelsysteem voor Engelse teksten, dat met behulp van een agent deze zinnen verteld. BEAT staat voor Behavior Expression Animation Toolkit en is een systeem waarbij de gebruiker tekst kan invoeren die vervolgens door een virtuele agent verteld wordt. Deze agent gebruikt de tekst om non-verbaal gedrag toe te voegen om zo het vertellen aantrekkelijker te maken.

Voor zover wij kunnen overzien, is BEAT de eerste specifiek hierop toegespitste implementatie. Veel andere onderzoeken naar gebaren bij het vertellen, gaan over het herkennen van deze gebaren door een computer en over het genereren van de gebaren in een gesprek. Beide situaties zijn erg verschillend. Het herkennen van gebaren heeft als achtergrond natuurlijk wel dezelfde gedragswetenschappelijke basis, maar is wel een ander onderwerp. Ook het gebruiken van gebaren door een agent is anders, aangezien de agent daar een duidelijk communicatief doel wil bereiken en hierdoor ook veel makkelijke toepasselijke gebaren kan bepalen.

BEAT maakt gebruikt van verschillende technieken om het non-verbale gedrag te bepalen aan de hand van een gegeven tekst. Belangrijke concepten hierbij zijn Part-of-Speech tagging, theme/rheme detectie en de algemene kennis van de wereld beschikbaar in een knowledge base. Deze knowledge base bestaat o.a. uit generieke concepten, waarvoor bepaalde generieke gebaren gedefinieerd zijn. Theme/rheme detectie slaat op het concept dat informatie in zinnen grofweg in twee delen opgedeeld kan worden, namelijk informatie die al wel bekend is bij de lezer op een geven moment en informatie die nieuw is. Hierbij staat het theme van een zin op de al bekende informatie en rheme op de nieuwe informatie.

Er zijn een aantal belangrijke concepten in BEAT, welke wij geprobeerd hebben over te nemen. Belangrijk hierin zijn de modulaire opbouw en het real-time werken van het systeem. Het BEAT systeem bestaat uit een serie aan elkaar gekoppelde modules die telkens opereren op de uitvoer van de vorige module en zo invoer bieden aan de volgende, waarbij er extra informatie over de zin wordt toegevoegd. Zo is de eerste module een tagging module, die de ingevoerde zinnen tagged en deze vervolgens doorgeeft aan de zogenaamde Behavior Suggestion module. Dit deel bepaald aan de hand van een knowledge base en bepaalde discourse modellen welke gedragingen toepasselijk zouden kunnen zijn in de huidige context van de zin. Hierna komt er een Behavior selector, deze bepaalt welk gedrag werkelijk uitgevoerd wordt. Vervolgens wordt de uitvoer hiervan daar een scheduler gestuurd, die de agent aanstuurt aan de hand van een woord timer.

Voor het genereren van gebaren gebruikt BEAT specifieke gebaar generatoren. Deze zijn opgedeeld in een aantal categorieen voor verschillende soorten gebaren. Zo is er binnen het een systeem o.a. een beat gesture generator, maar ook een gaze generator, een generator voor gebaren van verrassing en voor het knipperen met de ogen. Deze geven alle uitvoer die vervolgens geselecteerd kan worden door de gebaren selector van BEAT.

2.2 Virtual Storyteller

De Virtual Storyteller [12] is een systeem dat binnen HMI ontwikkeld is. De focus van dit systeem ligt op het zelf genereren van verhalen. het vertellen van de verhalen is een ondergeschoven kindje. Op dit moment wordt hiervoor Microsoft Agent gebruikt. De Microsoft Agent ondersteunt echter geen gebaren.

De focus van ons project ligt dus op een ander vlak, namelijk het vertellen zelf. Toch is de Virtual Storyteller een interessant systeem, juist omdat het zich op een totaal ander deel van sprookjes vertellen richt. Door dit verschil is het misschien mogelijk om de door ons ontwikkelde verteller te koppelen aan de reeds bestaande Virtual Storyteller.

Hoofdstuk 3

Design

In dit hoofdstuk wordt het ontwerp van ons systeem behandeld. Hieronder vallen belangrijke keuzen in het ontwerp en de mogelijkheden van het uiteindelijke systeem. Als eerste zullen we kort aangeven tot welk domein we het systeem beperken en waarom. Vervolgens zullen we bespreken welke gebaren we kunnen en willen implementeren in het systeem.

3.1 Domein

Gezien de beperkte duur van het project zagen we ons genoodzaakt om het systeem te beperken tot een behapbaar domein. Deze keuze kunnen we onderverdelen in twee deelkeuzes die we hieronder beschrijven.

3.1.1 Verhalenverteller

Na onze keuze voor het ontwerpen van een systeem dat gebaren gebruikt om communicatie te verbeteren, ligt het erg voor de hand om te kiezen voor een verhalenverteller. Dit biedt de mogelijkheid om simpele éénweg communicatie te gebruiken, waarbij verder niet veel rekening gehouden hoeft te worden met de luisteraar. De reden voor onze keuze voor dit type monoloog, ligt in het feit dat ook een menselijke verteller alle informatie over expressies en gebaren uit slechts de tekst moet halen. Bij een presentatie geven b.v. sheets veel extra informatie aan de luisteraar. Een andere overweging voor onze keuze is het idee dat dit systeem mogelijk een aanzet kan geven tot een andere vertellende agent binnen de Virtual Storyteller [12]. Dit lijkt ons een uitdagend idee. We weten niet in hoeverre dit allemaal mogelijk is in werkelijkheid, maar het is zeker een argument geweest.

3.1.2 Sprookjes

Om het domein van de verhalenverteller verder te beperken, hebben we ervoor gekozen om slechts een bepaald type verhalen te vertellen, namelijk sprookjes. Deze keuze hiervoor kan gemotiveerd door een aantal factoren. Belangrijk voor ons was de vaak expliciete aanwezigheid van acties en emoties binnen een sprookje. Een ander belangrijk argument is de simpele vertelvorm die vaak te vinden is binnen sprookjes. Hierbij moet niet alleen aan de structuur van zinnen gedacht worden, maar ook aan het woordgebruik. De vaak expliciet aanwezige emoties worden ook in vaak dezelfde termen verwoord. Belangrijk hierbij wel is dat er een knowledge base aangelegd zal moeten worden met dergelijke woorden en bijbehorende emoties. Verder zijn de grammaticale constructies meestal niet ontzettend complex en ook het citeren van mensen gebeurd op een simpele manier. Dit laatste zorgt voor het gemakkelijk kunnen implementeren van posture shifts indien een bepaalde actor in het verhaal geciteerd wordt. Verder sluit het vertellen van sprookjes ook weer aan op de Virtual Storyteller [12], om eventueel daarvoor als vertellende agent te dienen.

3.2 Mogelijke gebaren

Eén van de meest belangrijke aspecten bij het verhalen vertellen binnen ons systeem is het gebruik van gebaren om de tekst levendiger te maken. Hieronder zullen een aantal van deze mogelijke gebaren bekeken worden. Deze indeling is gebaseerd op het artikel Animated Conversation van Cassell en anderen [5].

3.2.1 Iconic gestures

Onder deze categorie vallen gebaren die illustreren wat de spreken zegt. Bij b.v. het woord *groot* wordt met de handen een wijd gebaar gemaakt, dat ook groot aanduidt. Deze gebaren zijn redelijk simpel te implementeren, het zal vooral gedaan worden door een keyword-spotter die aan bepaalde woorden een bepaald gebaar toekent.

3.2.2 Metaphoric gestures

In tegenstelling tot iconic gestures, geeft een metaphoric gesture niet direct het concept weer dat genoemd wordt. De handeling die gedaan wordt hoeft niet direct iets te maken te hebben met het gebaar, maar is wel illustratief daarvoor. Het voorbeeld dat dit idee duidelijk maakt, is het voorbeeld van Cassell, namelijk een met de hand terugtrekkend gebaar bij het uitspreken van de woorden *het opnemen van geld*. Een mogelijke manier om dit te implementeren is het gebruik van een knowledge base waarin bepaalde metaforen worden opgenomen om zo voor bepaalde termen een bepaalde metaforische actie te laten uitvoeren. Gezien het feit dat dit toch wel redelijk complex en tijdrovend is, zal in het prototype deze functionaliteit niet aanwezig zijn.

3.2.3 Deictic gestures

Deictic gestures zijn gebaren waarmee bepaalde objecten worden aangewezen. Indien men het heeft over *deze pan* terwijl men naar een bepaalde pan wijst, dan is dat een deictic gesture. Binnen een sprookje zijn dergelijke gebaren eigenlijk niet van toepassing. In het sprookje wordt alleen verwezen naar in dat sprookje aanwezige virtuele objecten en niet naar werkelijke objecten, wat het gebruik van deictic gestures overbodig maakt.

3.2.4 Beat gestures

Buiten de in de vorige secties gedefinieerde gebaren zijn er natuurlijk ook allerlei gebaren die niet direct op de inhoud van een verhaal slaan maar meer als communicatieve ondersteuning. Deze beat-gebaren willen wij natuurlijk ook binnen ons systeem opnemen, om zo een betere verhalenverteller te modelleren. We hebben hiervoor gekeken naar BEAT [4], waarin beats vooral op rhemes gelegd worden (oftewel op nieuwe informatie). Als simpele implementatie hebben we dan ook hiervoor gekozen, er vanuitgaand dat een lijdend voorwerp binnen een zin vaak nieuwe informatie inhoud, iets dat redelijk met de praktijk overeen komt. Voor meer informatie over hoe dit precies is geïmplementeerd, verwijzen we naar de bijbehorende sectie [5.4].

3.2.5 Posture shifts

Als laatste willen nog de door ons zelf gedefinieerde categorie posture shifts toevoegen. Alhoewel dit niet direct gebaren zijn, willen we deze toch noemen vanwege de significantie voor ons systeem. Door onze keuze voor het sprookjes domein, zal het in de verhalen vaak voorkomen dat bepaalde karakters door de verteller geciteerd worden. Op het moment dat menselijke vertellers dit binnen een verhaal tegenkomen, zal dat vaak duidelijk worden aangegeven door een posture-shift. Mensen veranderen hun balans, of draaien hun hoofd om zo aan te geven dat wat er dan volgt niet direct door hen gezegd wordt, maar dat ze enkel als spreekbuis dienen voor een karakter uit het verhaal. Hierbij komt het zelfs voor dat de verteller zijn stem aanpast aan het geciteerde karakter. Dit

blijkt onder andere uit onderzoek dat gedaan is door Cassell en Ryokai waarbij gekeken is naar het gebruik van vertelstemmen door kinderen [3].

Om dit te goed te implementeren willen we pronoun resolution gebruiken, zodat bij het citeren van bepaalde actors bepaald kan worden welke actor wat zegt. Vervolgens kan voor deze actor een andere pose aangenomen worden en kan zelfs een andere stem gebruikt worden.

3.3 Architectuur

De architectuur van het systeem is een zogenaamde pipe-and-filter architectuur. Dit houdt in dat het systeem uit verschillende modules bestaat, waarbij elke module bepaalde invoer en uitvoer heeft. Verschillende modules worden gekoppeld door de in- en uitvoer van bepaalde modules aan elkaar te koppelen.

Het grote voordeel van deze architectuur is de mogelijkheid om bepaalde modules te vervangen door andere, zodat het systeem makkelijk binnen andere systemen ingebouwd kan worden, bijvoorbeeld binnen een verhalen genereer systeem. Door de pipe-and-filter structuur kunnen ook parallel verschillende acties uitgevoerd worden. Een module die een bestand uitleest hoeft niet klaar te zijn voordat de PoS-tagger zijn werk kan doen.

In bijlage A kan een klassendiagram gevonden worden, dat de architectuur duidelijk weergeeft. Ook is er een sequence diagram te vinden dat weergeeft hoe een enkele zin binnen het systeem verwerkt wordt. In het tweede diagram is te zien hoe de verschillende modules na elkaar gestart worden, waarna zinnen in diezelfde volgorde door de modules heen gaan.

3.3.1 Modules

Binnen het systeem zijn er een aantal modules geïmplementeerd. Deze zijn te herkennen in het sequence diagram, waar de flow van het programma weergegeven worden. Hieronder zal kort worden besproken wat elk van deze modules doet.

- **FileReaderModule**
  Deze module leest een bestand in en stuurt deze inhoud vervolgens het systeem in. Deze opzet is zo gekozen dat er ook makkelijk een andere invoer gebruikt kan worden, het enige wat een dergelijke input moet doen is regels tekst naar het systeem sturen.

- **SentenceSplitterModule**
  In deze module worden de regels van het inkomende bestand opgesplitst in gehele zinnen. Hiervoor wordt gekeken naar een . in een zin, maar wordt ook rekening gehouden met constructies zoals *Hans zei: "Grietje, wat vind je van m'n jurkje."*.

- **POSTaggerModule**
  In deze module wordt de POSTagger van Rieks op den Akker [9] aangeroepen en wordt de uitvoer hiervan aan de volgende module doorgegeven.

- **TaggerModule**
  Deze module bevat de belangrijkste functionaliteit van het systeem. Deze module bestaat uit een aantal verschillende losse componenten:

  - **AlpinoTagger**
    In dit deel wordt de invoer naar de Alpino parser [1] gestuurd en wordt de invoer weer terug aan het systeem gegeven. Voor meer informatie over deze tagger zoals deze binnen deze module gebruikt wordt, zie [4.2].

  - **BEATTagger**
    De BEAT Tagger voegt aan de zin bepaalde BEAT gebaren toe, zoals deze in het gedeelte over de BEAT Tagger worden uitgelegd [5.4].

  - **KeywordSpotter**
    Dit deel van de tagger kijkt naar bepaalde keywords in de tekst en geeft hier bijhorende tags bij. Deze tags kunnen vervolgens door bijvoorbeeld de Living Actor module worden gebruikt om de tags te mappen op bepaaldegebaren die de Living Actor kan uitvoeren. Hierbij is te denken aan woorden die bepaalde emoties of handelingen representeren.
  - **QuoteTagger**
    In deze module wordt het citeren van bepaalde personen bepaald/bijgehouden. Dit wordt gebruikt voor de door ons gedefinieerd posture shifts en stem veranderingen bij het citeren van bepaalde actoren. Voor een uitgebreidere beschrijving hiervan verwijzen we naar de bijbehorende sectie onder het hoofdstuk Technologieën [5.3].

- **AgentController**
  In deze module wordt de uiteindelijke agent aangestuurd. In eerste instantie was het de bedoeling om de LivingActor agent alle gebaren te laten verzorgen, maar door allerlei problemen met LivingActor doet deze module nu niet veel meer dan de gegenereerde tags bij de bijbehorende zinnen als uitvoer geven.

Hoofdstuk 4

Technologieën

In dit hoofdstuk zullen we kort behandelen welke bestaande technologieën we gebruikt hebben bij de implementatie van ons systeem en waarom voor die bepaalde oplossing is gekozen.

4.1 PoS Tagging

Één van de meest basale technieken die nodig zijn voor het systeem dat wij voor ogen hebben, is het gebruiken van een PoS tagger. Een POS tagger kent een tag toe aan elk woord die aangeeft welk type woord het is. Bij woordtypen moeten we denken aan werkwoorden, zelfstandige naamwoorden enzovoorts. Door het gebruik van een dergelijke tagger is het makkelijker om de grammaticale structuur van de zinnen te achterhalen. Zo wordt het bijvoorbeeld makkelijker om het onderwerp of lijdend voorwerp van een zin te bepalen. Immers het onderwerp kan bijvoorbeeld niet bestaan uit een werkwoord, maar wel uit een zelfstandig naamwoord. De POS-tagger dient dan ook vooral als basis voor de andere technieken.

In ons systeem wordt de POS-tagger gebruikt voor de anaphora resolutie. Bij anaphora resolutie wordt bepaald naar welke entiteit een persoonlijke voornaamwoord verwijst. De POS-tagger kan bij de spreker detectie van dienst zijn om snel persoonlijke voornaamwoorden te vinden en de veelvoud en het geslacht hiervan te bepalen. Op deze manier kunnen we bepaalde mogelijkheden uitsluiten met betrekking tot de verwijzing. Zo kan het persoonlijke voornaamwoord 'wij' bijvoorbeeld niet naar een enkelvoudige referentie verwijzen. In het hoofdstuk `Implementatie5` zullen we verder uiteenzetten hoe het anaphora resolutie algoritme werkt. De POS-tagger die we gebruiken is ontwikkeld door Rieks op den Akker [9] voor de Nederlandse taal.

4.2 Alpino Parser

De Alpino Parser [1] is een bestaande parser voor Nederlandse zinnen, ontwikkeld aan de Rijksuniversiteit Groningen. Het kan een zin parsen en deze omzetten in een boomstructuur die de syntactische structuur van deze zin representeert. Deze techniek is nuttig voor o.a. het bepalen van het onderwerp in een zin en kan ook helpen bij theme/rheme detectie. Hoe dit precies in zijn werk gaat zullen we uiteenzetten in het hoofdstuk `Implementatie5`. Het bepalen van de theme/rheme is weer belangrijk om te weten bij het bepalen van de beats in de zin.

4.3 Living Actor

Een virtuele verhalenverteller stelt natuurlijk niks voor zonder een virtuele embodyment. Hiervoor zijn een aantal verschillende mogelijkheden beschikbaar. Zo zijn er de embodied conversational agents RUTH [7] en Greta [10]. Deze hebben echter als nadeel dat ze slechts uit een hoofd bestaan waardoor het maken van gebaren vrij onmogelijk wordt. Dit is jammer, omdat deze toolkits heel

wat uitgebreider en flexibeler lijken te zijn dan de toolkit die we uiteindelijk hebben gekozen. Verder maakt de BEAT toolkit [4] waarop ons systeem grotendeels gebaseerd is ook gebruik van een embodied conversational agent. We hebben echter niet kunnen achterhalen welke dit is.

Uiteindelijk hebben we gekozen voor Living Actor [2], vooral uit praktische overwegingen. Dit systeem biedt een simpele agent die door middel van simpele commando's bepaalde gebaren kan vertonen. Dit heeft echter wel als nadeel dat we slechts een beperkte set gebaren tot onze beschikking hebben. Toch hebben we hiervoor gekozen, zodat we snel een enigszins acceptabele visualisatie van ons systeem hebben. We zullen echter zien dat de daadwerkelijke visualisatie van het sprookje terzijde is geschoven ten gunste van het daadwerkelijk taggen van de tekst vanwege praktische overwegingen.

4.4 Microsoft Speech

Om onze virtuele verhalenverteller ook nog echt te kunnen laten vertellen, hebben we gebruik gemaakt van Microsoft Speech. De keuze hiervoor was simpel, Living Actor ondersteunt namelijk alleen deze TTS engine. De kwaliteit is niet echt om over naar huis te schrijven en intonaties en dergelijke zijn al helemaal niet mogelijk. In een beter systeem is dit echter wel wenselijk, want veel aspecten van het verhalen vertellen komen ook in veranderingen van de stem naar voren. Hierbij valt onder andere te denken aan het creëren van spanning door middel van de stem en het onderscheiden van personen door middel van verschillende stemmen.

Hoofdstuk 5

Implementatie

In dit hoofdstuk zullen we een aantal belangrijke aspecten van de implementatie van het systeem bespreken. Hierbij zullen een aantal technische ontwerpbeslissingen naar voren komen. Helaas zijn een aantal van deze ontwerpbeslissingen vanuit praktische overweging gemaakt in verband met de beperkte duur van het project.

5.1 PoS Tagging

De PoS-tagger die door ons gebruikt is, is een PoS-tagger voor het Nederlands ontwikkeld door Rieks op den Akker [9]. Deze was erg makkelijk binnen het systeem te integreren en werkte zonder moeilijkheden. De correctheid van de uitvoer laat echter nogal eens te wensen over. Meer hierover in het hoofdstuk `Evaluatie`6.

5.2 Alpino Parser

Aangezien de Alpino Parser een systeem is dat op Unix-achtige systemen draait, hebben we in eerste instantie geprobeerd het onder Cygwin aan de praat te krijgen. Aangezien Living Actor een Windows applicatie is, hebben we toen voor een client/server model gekozen om de Alpino Parser te kunnen gebruiken. We hebben een server opgezet waarop de Alpino Parser draait, deze kan door een simpele client binnen ons systeem aangeroepen worden, om zo toch de parser te kunnen gebruiken. De overhead van dit systeem is nagenoeg niet aanwezig, aangezien het parser van de zinnen zelf significant meer tijd vereist.

5.3 Quote tagger

Dit onderdeel van het systeem zorgt ervoor dat de citaties van karakters uit het sprookje op de juiste manier getagd worden. Dit betekent dat bij het begin van een citatie (de openende aanhalingstekens) een tag wordt toegevoegd om aan te geven dat een karakter begint te spreken. In het geval dat Hans bijvoorbeeld wat gaat zeggen zal een tag 'actor(Hans)' toegevoegd worden. Aan het einde van het citaat zal weer een tag toegevoegd worden om aan te geven dat de verteller weer zal spreken. In de uiteindelijke agent zullen deze tags vertaald moeten worden naar een geschikte verandering van pose. Eventueel kan zelfs de stem van de spraaksynthese aangepast worden. Dit valt echter niet onder de quote tagger die we hier beschrijven.

Om te bepalen welk karakter aan het woord is bij een citaat zullen we moeten bepalen welk karakter het onderwerp is van de omliggende zin. In de zin `Hans zegt: "Hallo."` is dit bijvoorbeeld duidelijk dat het karakter Hans de spreker is. Moeilijker wordt het als we persoonlijke voornaamwoorden tegenkomen. In de zin `Hij zegt: "Hallo."` is het bijvoorbeeld al een stuk

moeilijker om te bepalen wie de spreker is. Dit probleem staat bekend als anaphora of pronoun resolution.

In het boek Speech and Language Processing [6] van Jurafsky en Martin worden verschillende algoritmen besproken voor het pronoun resolution probleem. Wij hebben gekozen voor een implementatie gebaseerd op het eerst besproken algoritme. Dit algoritme is oorspronkelijk ontwikkeld door Lapin en Leass [11]. Dit algoritme is gebaseerd op een simpel score systeem dat rekening houd met hoe recent iets geweest is en ook met syntactische voorkeuren. Vanuit praktische overwegingen hebben we sommige delen van het algoritme niet geimplementeerd. Voor de overzichtelijkheid zullen we het algoritme zoals wij het geimplementeerd hebben uiteenzetten.

Het algoritme kan onderverdeeld worden in 2 operaties: het updaten van de salience factoren en het oplossen van referenties. We zullen beginnen met het updaten van de salience factoren. Salience is een term die gebruikt wordt om aan te geven in welke mate een entiteit geactiveerd is. De salience factoren zijn de factoren die de mate van activering bepalen en dus samen de salience van een enititeit bepalen. Elke keer als we een entiteit in de tekst tegenkomen in de vorm van een zelfstandig naamwoord of persoonlijk voornaamwoord moet de salience van die entiteit aangepast worden. Immers door de entiteit te noemen wordt deze in meer of mindere mate geactiveerd. Dit aanpassen doen we door te bepalen welke salience factoren van toepassing zijn. De factoren die we onderscheiden zijn:

Sentence recency	Toegekend de eerste keer dat een entiteit binnen een zin genoemd wordt.	100
Subject emphasis	Toegekend als het woord het onderwerp is van de huidige zin.	80
Existential emphasis	Is het een onderwerp van het existentiele soort (er is) ken dan slechts 70 punten toe.	70
Direct object emphasis	Toegekend als het woord het lijden voorwerp is.	50
Indirect object emphasis	Toegekend als het woord het meewerkend voorwerp is.	40
Non-adverbial emphasis	Toegekend als het woord niet binnen een prepositional phrase valt.	50
Head noun emphasis	Toegekend als het woord niet ingebed is in een grotere noun phrase.	80

Als de toe te passen factoren bepaald zijn worden de waardes opgeteld bij de huidige salience van de entiteit. Telkens als een zin eindigt worden de salience waarden van alle entiteiten gehalveerd. Het originele algoritme onderscheid nog twee salience factoren namelijk `role parallelism` en `cataphora` factoren. De eerste wordt toegepast als de referentie en referent dezelfde rol binnen de zin vervullen. De tweede als de referentie voor de referent voorkomt in de tekst. Deze factoren hebben respectievelijk weinig effect (role parallelism) en komen niet vaak voor (cataphora). Verder bleken deze factoren ook moeilijk te testen. Met name de cataphora aangezien ons systeem uitgaat van een pipe en filter architectuur. Dit betekent dat de zinnen van links naar rechts afgewerkt worden. We kunnen dus niet weten of er naar een entiteit later in de zin verwezen wordt. Om deze redenen hebben we deze laatste twee factoren weg gelaten.

De tweede operatie van het algoritme is het daadwerkelijk oplossen van de referenties. Oftewel bepalen naar welke entiteit een referentie refereert. Ons algoritme negeert meervoudige persoonlijke voornaamwoorden. Dit is mogelijk, omdat we uit gaan van de aanname dat er altijd slechts één persoon tegelijk geciteerd kan worden. Verder hebben we ook niet gekeken naar het geslacht van de persoonlijke voornaamwoorden. Hiervoor konden we simpelweg niet genoeg informatie uit de POS tagger of Alpino parser krijgen. De operatie komt dan op het volgende neer.

1. Neem de lijst van alle entiteiten tegengekomen tot nu toe.
2. Als we aan het citeren en de referentie is in de eerste persoon, verwijder alle entiteiten behalve de spreker.
3. Als we aan het citeren en de referentie is niet in de eerste persoon, verwijder de spreker uit de lijst.

4. Kies de entiteit uit de lijst met de hoogste salience.

Voor het opslaan van de referenten die we zijn tegengekomen hebben we een interne `Actor` klasse gemaakt. Ook deze `Actor` klasse is enigszins gesimplificeerd ten opzichte van het oorspronkelijke algoritme. We houden namelijk niet alle frasen bij die referen naar een referent. Bij het bepalen van de juiste referent in het geval van zelfstandige naamwoorden gaan we alleen uit van het zelfstandige naamwoord zelf. We negeren de bijbehorende bijvoeglijke naamwoorden dus.

Als eerste implementatie gebruikten we de POS tags om te bepalen of een woord een zelfstandig naamwoord of een persoonlijk voornaamwoord was. Het bleek echter dat de POS tagger nogal eens een zin van verkeerde POS tags voorzag. Dit had bijvoorbeeld als gevolg dat interpunctie tekens als referenten werden benoemd. Om dit te verbeteren hebben we de POS tags uit de Alpino parser gehaald. Deze bleken heel wat betrouwbaarder te zijn. Voor het bepalen van de juiste salience factoren hebben we ook de tags van de Alpino Parser gebruikt.

5.4 BEAT Tagger

Voor het taggen van de beat gebaren hebben we eerst geprobeerd een goed algoritme te vinden. Een veelbelovend artikel van Eva Hajicova et al [8] beschrijft een algoritme om de topic en focus van een zin te bepalen. Deze komen grofweg overeen met de theme en rheme. Dit algoritme zou zeer handig zijn voor het bepalen van de beat gebaren ware het niet dat het algoritme te complex is om binnen een redelijk tijdsbestek te implementeren. De uiteindelijke implementatie van het genereren van beats is dan ook simpel gebleven.

Omdat het bepalen van de theme en rheme buiten de mogelijkheden viel hebben we gekozen voor een ruwe benadering van het concept. Zoals gezegd bestaat de rheme van een zin uit dat wat nieuw geïntroduceerd wordt en de theme uit wat reeds geïntroduceerd is. Ter benadering hiervan hebben we aangenomen dat het onderwerp van een zin de theme vormt en het lijdend voorwerp de rheme. Naar onze mening zal deze benadering in de meeste gevallen kloppen, vooral in het sprookjesdomein. Gezien deze aanname moeten we bepalen wat het onderwerp en lijdend voorwerp van een zin zijn. Hiervoor hebben we gebruik gemaakt van de Alpino Tagger. Deze levert niet altijd correcte resultaten op, maar was de beste optie die we binnen redelijke tijd voor handen hadden.

De implementatie van de BEAT Tagger definieert drie beat-levels. Hoe hoger de beat-level hoe meer articulatie we toevoegen aan de tekst. Level 1 bestaat uit head-nouns binnen een block welke Alpino als lijdend voorwerp (obj1) heeft getagged. Level 2 bestaat uit de head-nouns van meewerkende voorwerpen (obj2). Level 3 bestaat uit de head-nouns van het onderwerp. De tagger tagged altijd alle levels, zodat in een later stadium bepaald kan worden welke beats daadwerkelijk gebruikt gaan worden voor de actor. Het idee is dat in een later stadium gebaren worden gegenereerd, de beats geven alleen de plekken aan waar de gebaren eventueel zouden kunnen komen. Het bepalen van welke gebaren op de plek van de beats komt kan willekeurig. De levels bieden een interface voor de mate van expressiviteit van de agent. Een minder expressieve agent maakt alleen gebaren op de plekken van beat-level 1. Een meer expressieve agent maakt gebruik van alle levels. De uiteindelijke implementatie biedt nog geen mogelijkheid om deze mate van expressiviteit in te stellen, omdat we de visualisatie achterwege hebben gelaten.

5.5 Keyword Tagger

De laastste module die gebaren tags toevoegt aan de tekst naast de quote en beat tagger is de keyword tagger. Deze module is in vergelijking met de voornoemde modules zeer simpel in zijn implementatie. Kort gezegd kan in een configuratiebestand aangegeven worden welke woorden er getagd moeten worden en welke tag ze daarvoor moeten krijgen. De tag geeft aan welk soort gebaar er gemaakt moet worden. Momenteel is het aantal gespecificeerde keywords zeer laag. Dit aantal is echter zeer makkelijk uit te breiden door het configuratiebestand aan te passen. Een gebrek is wel dat de keyword tagger geen rekening houdt met de verschillende vormen van een

woord (inflectie). Het blijkt dat de alpino tagger wel voorziet in een stam vorm voor de woorden, maar we zijn er niet aan toegekomen om hiervan gebruik te maken.

5.6 Living Actor

Voor de agent die het uiteindelijke sprookje zou moeten vertellen hadden we zoals gezegd gekozen voor de Living Actor toolkit. De vorm waarin Living Actor het meest gebruikt wordt, is als ActiveX object binnen een webpagina die via Microsoft Internet Explorer opgevraagd worden. Aangezien wij in onze implementatie van Java uitgegaan zijn, hebben we hiervoor een oplossing gezocht. Er was een pakket waarmee Living Actor via allerlei omwegen binnen Java aangeroepen kan worden, iets dat we uiteindelijk aan de praat hebben gekregen.

De module die de Living Actor bestuurt is verantwoordelijk voor het vertalen van de tags naar de juiste gebarencommando's voor de Living Actor. Natuurlijk willen we dat dit vertalen flexibel was en dus gemakkelijk uitgebreid kon worden. Dit hebben we verwezenlijkt door in een extern bestand een lijst te maken met de tags en hun bijbehorende vertaling. De vertaling kan bestaan uit een gebaar of een gezichtsuitdrukking of een combinatie hiervan. Voor elke tag is echter slechts 1 vertaling mogelijk. De `AgentController` module leest dit bestand uit en gebruikt het voor de daadwerkelijke vertaling van de tags.

Aangezien we aan het einde van het project in tijdnood kwamen hebben we de agentcontroller op een laag pitje gezet. Het leek ons belangrijker om de tagging zover mogelijk af te krijgen. Het bleek bovendien zeer lastig te zijn om de Living Actor stil te laten staan. De Living Actor veranderde zeer vaak van positie zonder dat we dat wilden. Helaas hebben we geen redelijke manier kunnen vinden om dit constante wisselen van pose te onderdrukken. Als laatste bleek ook de set van gebaren van de Living Actor niet afdoende te zijn voor onze wensen. De keuze aan gebaren was simpelweg te beperkt om een redelijke vertaling voor alle mogelijke tags te vinden.

Om deze redenen hebben we ervoor gekozen om de implementatie van de Living Actor niet verder uit te werken. Samengevat is de Living Actor niet flexibel genoeg voor onze wensen. We zullen in het hoofdstuk *Toekomstig Werk* [7] bespreken wat we verwachten van een goede agent voor de visualisatie.

Hoofdstuk 6

Evaluatie

In dit hoofdstuk zullen we een aantal problemen behandelen die we niet echt hebben kunnen oplossen. Verder zullen we reflecteren op het uiteindelijke systeem. De technische problemen bestaan vooral uit problemen met externe software die wij gebruikt hebben. In de reflectie zullen we ingaan op welke aspecten we wel en niet tevreden zijn over ons systeem.

6.1 Overzicht functionaliteit prototype

- **Pos tagger**
  In het systeem wordt intern gebruik gemaakt van twee postaggers. De eerste postagger is die van Rieks op den Akker. De tweede postagger is de Alpino tagger. Deze tagged onder andere het onderwerp en lijdend voorwerp. In de laatste implementatie wordt er alleen nog maar gebruik gemaakt van de Alpino tagger. Wel worden de woorden ook met de tagger van Rieks getagged maar niet gebruikt.

- **Keyword spotter (Iconic gestures)**
  De keyword spotter voegt tags toe aan de hand van een database. Deze database is momenteel toegespitst op het sprookje Hans en Grietje. Dit zijn tags voor zowel gebaren als emoties. De huidige database bevat de tags GROOT, SOMBER, PIEKER, ZUCHT, VRAAG, KOKEN, VER, ZUCHT en VERDRIET.

- **Quote tagger (Posture shifts)**
  De quote tagger tagged wanneer er een andere actor aan het woord komt. Hierdoor kunnen posture shifts en spraakwisseling geïmplementeerd worden. De quote tagger voegt tags toe als ACTOR(hans) waarbij de actor hans uit de tekst wordt gehaald. De algemene verteller wordt ACTOR(fatt) genoemd.

- **Beat tagger (Beat gestures)**
  De beat tagger tagged zelfstandige naamwoorden op de manier zoals die is beschreven in 5.4 De tags die de beat tagger toevoegd zijn de tags BEAT1, BEAT2 en BEAT3.

- **Living actor**
  De living actor implementatie is in het huidige prototype helaas achtergebleven in verband met de tijdsduur van het project. De implementatie is niet volledig en de focus is gelegd op het taggen van pure tekst zodat de implementatie van de Living actor eventueel later nog eenvoudig geperfectioneerd en uitgebreid kan worden.

Deictic gestures en metaphoric gestures zijn niet geïmplementeerd zoals beschreven staat in 3.2.

6.2 Technische Problemen

In deze sectie zullen we kort een aantal problemen beschrijven die we tegen zijn gekomen bij het gebruiken van technologie van derden. Belangrijke problemen zijn de volgende:

- **PoS-Tagger**
  Bij de PoS-Tagger kwamen we na een tijdje gebruik problemen tegen bij de wat complexere zinnen. Bij de volgende twee zinnen b.v. *Hans liep door een bos met bomen. Plotseling ziet hij een grote eikel liggen.* Bij deze zinnen wordt bij de tweede zin de afsluitende punt als een noun gezien en zijn ook de andere tags niet betrouwbaar. Als oplossing hiervoor hebben we ook de PoS-tags gebruikt die Alpino oplevert, deze leken betrouwbaarder te zijn. De reden dat de POS-tagger in de fout gaat bij deze zinnen is ons niet duidelijk. Wellicht heeft het te maken met de implementatie van de parser binnen de POS-tagger.

- **Alpino Parser**
  Ook bij de Alpino parser kwamen we een aantal problemen tegen. Het belangrijkste probleem hier was het parsen van zinnen waarin mensen geciteerd worden. Bij de zin *"Grietje, wat vind je van m'n mooie jurkje", zei Hans*, wordt Hans niet goed als het onderwerp van de zin gedetecteerd. Dit zorgt weer voor vervelende problemen bij o.a. de anaphora resolution.

- **Living Actor**
  Er is soms het probleem dat de Living Actor niet de gebaren maakt die wij hem opdragen. We hebben hier verder geen oplossing voor kunnen vinden, aangezien het met een herstart van het systeem soms weer ineens wel werkte. Ook hebben we er verder weinig aandacht aan besteed, gezien onze focus op het taggen. Omdat de Living Actor zoveel problemen gaf hebben wij er voor gekozen om onze focus meer te verleggen naar het taggen van de text. Zo kan met het wijzigen van alleen de Actor module een andere actor worden aangestuurd.

6.3 Resultaten

In de appendix [B] zijn de resultaten te vinden van een aantal zinnen die door het systeem getagged zijn. Hierin zijn de beat gebaren weergegeven, maar ook het verwisselen van actoren door middel van quotes. Zoals te zien is zijn de aangegeven beats voorstelbaar indien het door een menselijke spreker uitgesproken worden. Aangezien de BEAT tags niets van doen hebben met keywords en alleen gebaseerd zijn op de structuur van de zin is deze methode algemeen toepasbaar.

Dit is uiteraard niet het geval bij het genereren van tags voor gebaren aan de hand van keywords. De gebruikte zinnen zijn specifiek voor het sprookje waarmee we het systeem hebben getest. Het systeem is nog slechts een prototype. Wel kan het systeem eenvoudig uitgebreid worden door nieuwe keywords toe te voegen. Dit maakt het systeem algemener toepasbaar.

De keyword spotter werkt door het herkennen van hele woorden. Beter zou zijn geweest om bij werkwoorden de stam van een woord te nemen. Daarmee voorkom je het probleem dat je voor elk werkwoord alle vervoegingen moet opgeven (inflectie). De Alpino Parser levert een attribuut "root" wat de stam voorstelt. Dit is dus eenvoudig te implementeren, helaas hadden wij hier niet meer genoeg tijd voor.

Verder blijkt dat er bij de meeste zinnen beats worden getagged, ondanks het feit dat er alleen op zelfstandige naamwoorden is gefocust omdat we nadruk willen leggen op het onderwerp en lijdend voorwerp. Bij verschillende woorden zijn beats aangegeven volgens de expressiviteit levels die eerder al behandeld zijn [5.4].

Het systeem is niet getest met andere sprookjes, hier zijn wij niet aan toegekomen. Verwacht wordt dat de BEAT tags net zo goed zullen werken bij andere sprookjes als bij Hans en Grietje. Uiteraard zullen andere sprookjes nieuwe woorden bevatten die interessant zijn voor de keyword spotter. De database met keywords zal dan aan deze sprookjes moeten worden aangepast. Als dit gedaan wordt met meerdere sprookjes dan wordt de database groter en zullen steeds meer tags herkend worden.

De reden dat wij ons hebben beperkt tot het sprookjes domein is voornamelijk gebaseerd op het feit dat sprookjes vaak specifiek worden verteld, in makkelijke zinnen 3.1. Daarom werkt keyword spotting goed. Het systeem zou ook kunnen worden gebruikt voor gewone verhalen, waarschijnlijk werken de BEATS ook dan goed, mits de structuur van de zinnen niet al te complex is en de POS-taggers deze zinnen ook goed kunnen parsen. Ook keyword spotting kan voor gewone verhalen werken alleen worden in verhalen meer metaforen gebruikt. De betekenis van een metafoor achterhalen is een complex proces. Daarnaast wordt in gewone verhalen waarschijnlijk minder vaak geciteerde tekst gebruikt, waardoor actor wisselingen niet vaak plaatsvinden.

6.4 Conclusie

Het uiteindelijke systeem werkt naar onze mening redelijk goed. Er worden BEAT tags gegenereerd op naar ons inzicht logische plaatsen. Het systeem heeft echter nog wel duidelijk de tekortkomingen van een in snelle tijd ontwikkeld prototype. We hebben geen fatsoenlijk grote database voor de keyword spotter, deze is nu toegespitst op het sprookje van Hans en Grietje en moet worden uitgebreid om algemeen toepasbaar te zijn. Daarnaast is de code lang niet altijd even netjes. Dit komt vooral omdat we zo snel mogelijk het systeem werkend wilden hebben. De hoeveelheid informatie die we op dit moment aan zinnen en woorden kunnen toevoegen is echter al behoorlijk uitgebreid. Ook het systeem waarmee we detecteren wie aan het woord is werkt behoorlijk.

Een ander nadeel is de nogal grote afhankelijkheid op de goede uitvoer van de PoS-tagger en de Alpino parser. Indien deze foute resultaten opleveren, kan ons systeem deze fouten niet herstellen.

Hoofdstuk 7

Toekomstig Werk

In dit hoofdstuk zullen we een aantal belangrijke punten behandelen waarvan wij vinden dat die niet mogen ontbreken, maar waarvoor er ook binnen het tijdsbestek van het vak geen tijd was. Als eerste zullen we het hebben over de visualisatie van het sprookje. Vervolgens zullen we het hebben over een meer gestructureerde manier van data doorvoeren. Daarna zullen we bespreken wat er verder nog onderzocht kan worden aan gebaren binnen het sprookjes domein.

7.1 Keyword spotting

De database van de keyword spotter is nu slechts gemaakt voor het sprookje Hans en Grietje. Deze moet om algemeen toepasbaar te zijn veel meer worden uitgebreid. Daarnaast is het interessant om niet gebruik te maken van specifieke woorden en vervoegingen van werkwoorden maar van de stam van een woord. Een voorbeeld hiervan is bijvoorbeeld de stam *zucht*. Dan worden vervoegingen als *zuchten, zuchtte etc.* automatisch toegevoegd.

7.2 Visualisatie

We hebben ervoor gekozen om zo snel mogelijk een werkende versie van ons systeem te krijgen. Dit in verband met de beperkte tijd van het project. Vandaar dat we voor de Living Actor toolkit gekozen hebben. Deze is echter wel erg beperkt gebleken. De annotatie van de tekst die door ons gedaan wordt, is al complexer dan door Living Actor weergegeven kan worden. Indien het systeem ergens anders gebruikt gaat worden, is het dan ook zeer aan te bevelen om een andere agent te gebruiken die een uitgebreider repertoire aan gebaren heeft. Nog beter zou zijn om zelf gebaren te kunnen kunnen definiëren. Daarbij is het niet praktisch dat de agent zoals bij Living Actor uit zichzelf gaat bewegen en gebaren. Dit willen we immers zelf controleren. Hierbij is het ook belangrijk dat de spraak goed gesynchroniseerd word met de gebaren. Als laatste is het wenselijk dat de agent gemakkelijk vanuit een java programma te besturen is. Uiteraard hebben we hier naar gezocht, maar we hebben geen niet-commerciele agents gevonden die aan deze wensen voldoen.

7.3 Gestructureerde Data

In dit project hebben we gebruik gemaakt van de string als principiële datatype voor het doorsturen van de data. De reden hiervoor was dat we een systeem wouden bouwen waar de data gemakkelijk doorheen gestreamd kon worden. Echter hoe meer we het systeem uitbouwden hoe minder bruikbaar dit systeem bleek. Zouden wij het systeem overnieuw maken, dan zouden we waarschijnlijk kiezen voor XML strings, zodat we hier gemakkelijke boom-structuren van kunnen maken. Een dergelijke boom-structuur leek op het einde vaak erg handig binnen de implementatie.

Een boom structuur is nuttig omdat dan per node informatie kan worden opgeslagen. Tevens kan data (in dit geval woorden) geordend worden. Het zingedeelte “Het mooie huis” wordt dan bijvoorbeeld samengevat onder een node “subject”. Op deze manier is het eenvoudiger om informatie toe te voegen, zonder dat andere modulen daar last van hebben.

7.4 Modelleren menselijke vertellen

Voor zover wij het wisten te vinden, zijn er niet echt onderzoeken uitgevoerd die kijken naar het menselijk vertelgedrag. Hiermee bedoelen we het specifieke gedrag waarbij een enkel persoon een voordracht houdt. Er is echter al wel veel onderzoek gedaan naar het meer algemene concept van communiceren door mensen. Dergelijke onderzoeken zien we ook vaak terug komen als referenties bij andere artikelen. Het zou ons interessant lijken als er een keer onderzocht werd of het vertellen van verhalen niet resulteert in bijvoorbeeld expressiever gebarengebruik of zelfs in ander gebruik. Dit onderzoek zou dan b.v. inhouden dat een aantal personen uitgebreid geobserveerd worden tijdens het vertellen van een verhaal. Met behulp van deze data kan dan gekeken worden of dit verschil oplevert in de manier van vertellen ten opzichte van de bestaande, vaak op conversaties gebaseerde, literatuur. Dit geheel is natuurlijk een heel een onderzoek op zich, iets waar binnen dit project geen ruimte voor was.

7.5 Uitgebreidere Tagging

Op dit moment proberen we zo veel mogelijk informatie binnen een tekst te krijgen, maar dit kan natuurlijk altijd nog uitgebreid worden met meer nuttige informatie. Zo zou het bijvoorbeeld interessant zijn om woorden te kunnen taggen op intonatie. Denk aan het benadrukken van bepaalde woorden door ze harder, langer of hoger uit te spreken. Een andere mogelijkheid is het toevoegen van metaforische gebaren. Ook hiervoor zou een database moeten worden aangelegd en moet het systeem kijken wanneer deze gebaren toepasselijk zijn.

Bibliografie

[1] Gosse Bouma and Gertjan van Noord. Alpino: Automatisch ontleden van het nederlands. `http://odur.let.rug.nl/~vannoord/alp/Alpino/`.

[2] Cantoche. Living actor. `http://www.livingactor.com`.

[3] J. Cassell and K. Ryokai. Making space for voice: Technologies to support children's fantasy and storytelling. *Personal Ubiquitous Comput.*, 5(3):169–190, 2001.

[4] J. Cassell, H. Vilhjálmsson, and T. Bickmore. Beat: the behavior expression animation toolkit. *Proceedings of SIGGRAPH 2001*, 2001.

[5] Justine Cassell, Catherine Pelachaud, Norman Badler, Mark Steedman, Brett Achorn, Tripp Becket, Brett Douville, Scott Prevost, and Matthew Stone. Animated conversation: rule-based generation of facial expression, gesture & spoken intonation for multiple conversational agents. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 413–420, New York, NY, USA, 1994. ACM Press.

[6] Jurafsky D. and Martin J.H. *Speech and Language Processing*. Prentice Hall, international edition edition, 2000.

[7] Doug Decarlo and Matthew Stone. Rutgers university talking head. `http://www.cs.rutgers.edu/~village/ruth/`.

[8] Eva Hajicová, Petr Sgall, and Hana Skoumalová. An automatic procedure for topic-focus identification. *Comput. Linguist.*, 21(1):81–94, 1995.

[9] Rieks op den Akker. Pos-tagger. `http://wwwhome.cs.utwente.nl/~infrieks/`.

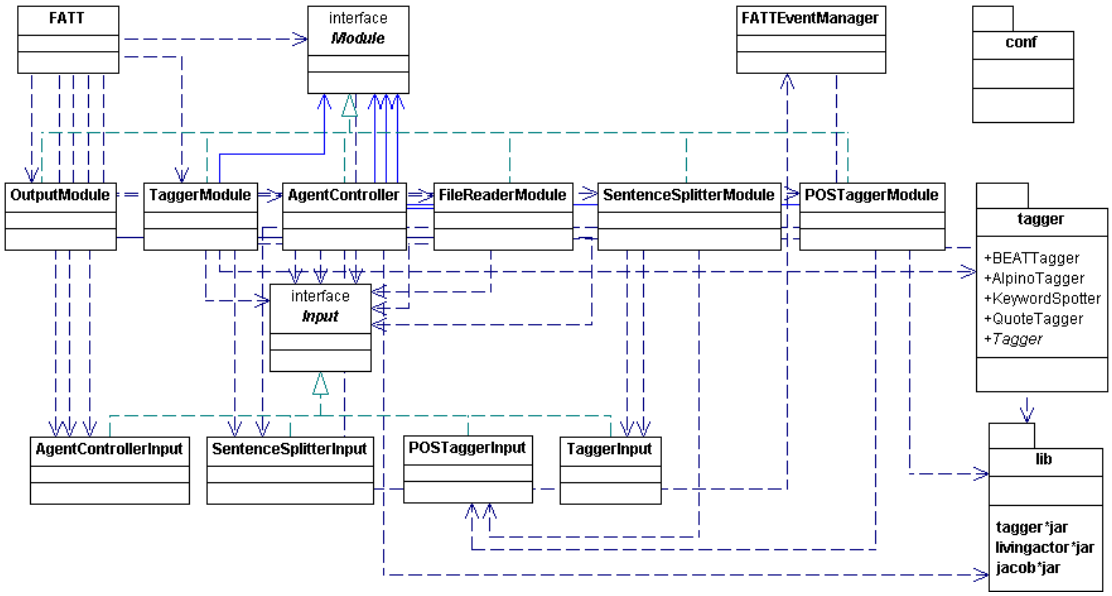
[10] S. Pasquariello and C. Pelachaud. Greta: A simple facial animation engine. *6th Online World Conference on Soft Computing in Industrial Appications*, 2001.

[11] Lappin S. and Leass H. An algorythm for pronominal anaphora resolution. *Computational Linguistics*, 4(20):535–561, 1994.

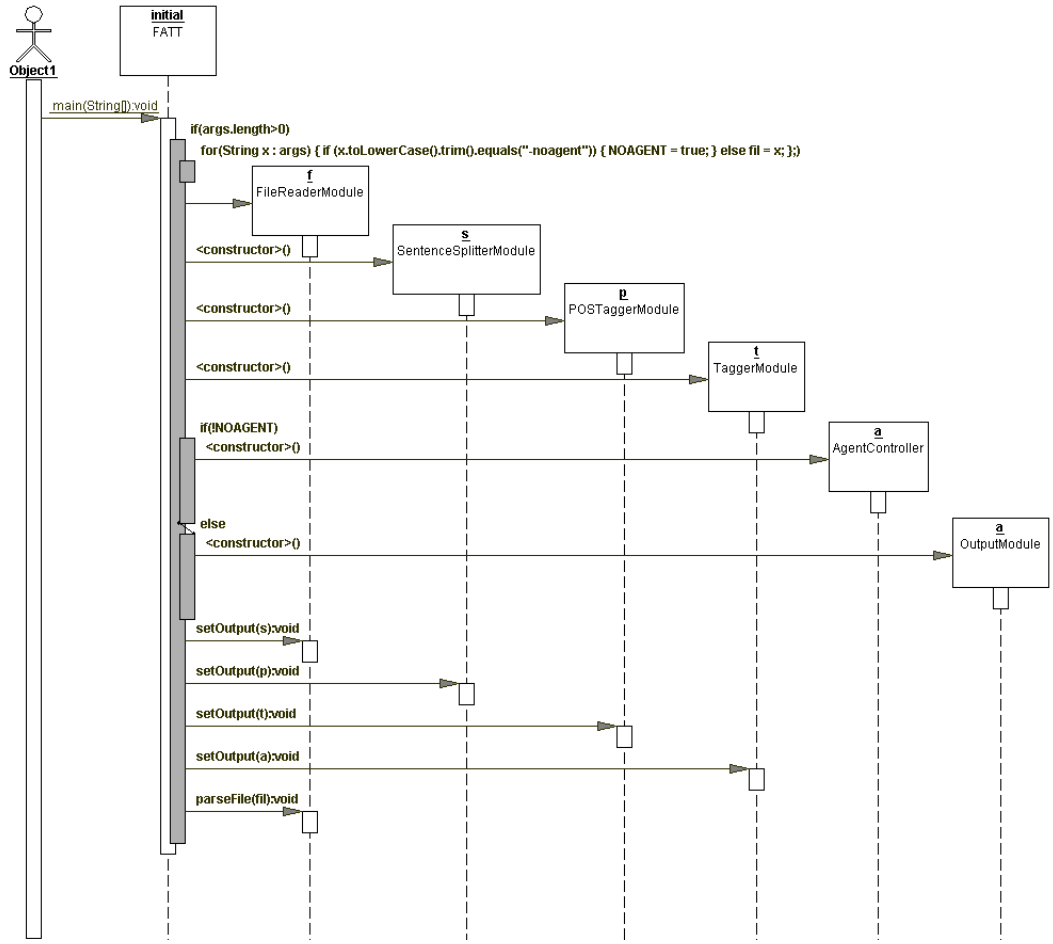
[12] Mariët Theune, Sander Faas, Anton Nijholt, and Dirk Heylen. The virtual storyteller: Story creation by intelligent agents. *Proceedings TIDSE 2003: Technologies for Interactive Digital Storytelling and Entertainment*, 2003.

Bijlage A

Diagrammen



Figuur A.1: Klassendiagram



Figuur A.2: Sequencediagram

Bijlage B

Resultaten

In deze bijlage wordt respectievelijk de werking van de keyword spotter, quote tagger en beat tagger laten zien. Per voorbeeld zin worden de tags die door het systeem zijn gegenereerd beschreven.

B.1 Keyword Spotting

De keyword spotter herkend woorden en voegt een tag toe die vervolgens herkend en gemapped kan worden op een gebaar.

- Ze liepen heel diep/VER het bos/BEAT1 in .

  In deze zin heeft de keyword spotter het woord diep herkend. In de database staat dat voor diep de tag VER gebruikt kan worden. De living actor kan VER vervolgens mappen op het gebaar "GestureBack". Daarnaast is door de beat tagger het woord bos getagged. Dit zal later worden beschreven.

  Ze/Pron(per,3,ev_of_mv,nom)/su,POSpron
  liepen/V(intrans,ovt,1_of_2_of_3,mv)/hd,POSverb
  heel/Adv(gew,geen_func,stell,onverv)/mod,POSadj
  diep/Adj(adv,stell,onverv)/hd,POSadj/VER
  het/Art(bep,onzijd,neut)/BEGINnp,BEGINobj1,det,POSdet
  bos/N(soort,ev,neut)/EINDnp,EINDobj1,hd,POSnoun/BEAT1
  in/Adv(deel_v)/svp,POSpart
  ./Punc(punt)/

- Eindelijk hadden ze wat te eten/KOKEN .

  De keyword spotter heeft het woord eten herkend en de tag KOKEN toegevoegd. De living actor kan deze tag mappen op het gebaar "Cook".

  Eindelijk/Adv(gew,geen_func,stell,onverv)/mod,POSadj
  hadden/V(hulp,ovt,1_of_2_of_3,mv)/hd,POSverb
  ze/Pron(per,3,ev_of_mv,nom)/su,POSpron
  wat/Pron(onbep,neut,zelfst)/mod,POSadv
  te/Prep(voor_inf)/hd,POSverb
  eten/V(trans,inf)//KOKEN
  ./Punc(punt)/

B.2 Quote tagger

De Qoute tagger houdt bij wie er aan het spreken is, zodat een actor bijvoorbeeld een andere stem kan kiezen wanneer een bepaald persoon spreekt.

- "/ACTOR(moeder) We moeten naar het bos/BEAT1 om hout/BEAT1 te hakken , "/ACTOR(fatt) zei hun moeder/BEAT3 .

  De Quote tagger heeft herkend dat de zin binnen de quotes uitgesproken wordt door "moeder". Er wordt de tag ACTOR(moeder) toegevoegd. De tag ACTOR(fatt) bij de sluitende quote betekent dat de verhalenverteller weer aan het woord is.

  "/Punc(aanhaal_dubb)//ACTOR(moeder)
  We/Pron(per,1,mv,nom)/su,POSpron
  moeten/V(hulp,ott,1_of_2_of_3,mv)/hd,POSverb
  naar/Prep(voor)/BEGINpp,BEGINld,hd,POSprep
  het/Art(bep,onzijd,neut)/BEGINnp,BEGINobj1,det,POSdet
  bos/N(soort,ev,neut)/EINDpp,EINDld,EINDnp,EINDobj1,hd,POSnoun/BEAT1
  om/Prep(voor)/cmp,POScomp hout/N(soort,ev,neut)/obj1,POSnoun/BEAT1
  te/Prep(voor_inf)/hd,POSverb
  hakken/V(trans,inf)/ ,/Punc(komma)/
  "/Punc(aanhaal_dubb)//ACTOR(fatt)
  zei/V(trans,ovt,1_of_2_of_3,ev)/hd,POSverb
  hun/Pron(bez,3,mv,neut,attr)/BEGINnp,BEGINsu,det,POSdet
  moeder/N(soort,ev,neut)/EINDnp,EINDsu,hd,POSnoun/BEAT3
  ./Punc(punt)/

- "/ACTOR(ouders) Ja , "/ACTOR(fatt) zeiden Hans en Grietje , "/ACTOR(ouders) we zijn er nog. "/ACTOR(fatt)

  Hier is een voorbeeld waarin het aangeven van de actor niet goed gaat. Het salience algoritme geeft aan dat de ouders spreken, terwijl dit Hans en Grietje zouden moeten zijn. Hierdoor wordt foutief de ACTOR(ouders) tag toegevoegd. Ouders wordt herkend uit vorige zinnen.

  "/Punc(aanhaal_dubb)//ACTOR(ouders)
  Ja/Int/tag,POStag ,/Punc(komma)/
  "/Punc(aanhaal_dubb)//ACTOR(fatt)
  zeiden/V(trans,ovt,1_of_2_of_3,mv)/hd,POSverb
  Hans/N(eigen,ev,neut)/BEGINconj,BEGINsu,cnj,POSname
  en/Conj(neven)/crd,POSvg
  Grietje/N(eigen,ev,neut)/EINDconj,EINDsu,cnj,POSname
  ,/Punc(komma)/ "/Punc(aanhaal_dubb)//ACTOR(ouders)
  we/Pron(per,1,mv,nom)/su,POSpron
  zijn/V(hulp_of_kopp,ott,1_of_2_of_3,mv)/hd,POSverb
  er/Adv(pron,er)/mod,POSadv
  nog./V(trans,verl_dw,onverv)/predc,POSnoun
  "/Misc(vreemd)//ACTOR(fatt)

B.3 BEATS

De BEAT tagger is verantwoordelijk voor het aangeven van BEATS.

- Ze hoorden alles/BEAT1 wat hun ouders/BEAT3 zeiden .

  "Alles" is in dit geval getagged met BEAT1 omdat dit een "hd" noun binnen een BEGINobj1 en EINDobj1 is. Een lijdend voorwerp dus. "Ouders" is in dit geval getagged met BEAT3

omdat dit een “hd” noun binnen een BEGINsu en EINDsu is. De tagger vindt dat ouders in dit geval het onderwerp is. “Ze” is niet getagged omdat dit geen POSnoun is (volgens de pos-tagger). Zie de tags hieronder.

Ze/Pron(per,3,ev_of_mv,nom)/su,POSpron
hoorden/V(trans,ovt,1_of_2_of_3,mv)/hd,POSverb
alles/Pron(onbep,neut,zelfst)/BEGINnp,BEGINobj1,hd,POSnoun/BEAT1
wat/Pron(betr,neut,zelfst)/rhd,POSpron
hun/Pron(bez,3,mv,neut,attr)/BEGINnp,BEGINsu,det,POSdet
ouders/N(soort,mv,neut)/EINDnp,EINDsu,hd,POSnoun/BEAT3
zeiden/V(trans,ovt,1_of_2_of_3,mv)/EINDnp,EINDobj1,hd,POSverb
./Punc(punt)/

- Daar maakte de houthakker/BEAT3 een vuur/BEAT1 .

  “Een vuur is” getagged met BEAT1 omdat dit de “hd noun” binnen een BEGINobj1 en EINDobj1 is. Een lijdend voorwerp dus. “De houthakker” is getagged met BEAT3 omdat dit de “hd” noun binnen een BEGINsu en EINDsu is. Een onderwerp.

  Daar/Adv(pron,aanw)/mod,POSadv
  maakte/V(trans,ovt,1_of_2_of_3,ev)/hd,POSverb
  de/N(soort,ev,neut)/BEGINnp,BEGINsu,det,POSdet
  houthakker/V(trans,conj)/EINDnp,EINDsu,hd,POSnoun/BEAT3
  een/N(eigen,ev,neut)/BEGINnp,BEGINobj1,det,POSdet
  vuur/N(soort,ev,neut)/EINDnp,EINDobj1,hd,POSnoun/BEAT1
  ./Misc(vreemd)/