Carrie the Cartoonist

Steven te Brinke (s0093122),
Marieke Franssen (s0022047),
Henk Erik van der Hoek (s0098396),
Brend Wanders (s0088897)

Begeleiders:
Mariët Theune,
Ivo Swartjes

5 maart 2008

Inhoudsopgave

Hoofdstuk 1

Inleiding

De vakgroep Human Media Interaction van de Universiteit Twente is al enige tijd bezig met onderzoek naar het automatisch genereren van verhalen. Dit onderzoek vindt plaats in het kader van het Virtual Storyteller project. Verhalen die door de Virtual Storyteller worden gegenereerd, worden momenteel weergegeven in een RDF formaat. Er zouden verschillende methoden gebruikt kunnen worden om zo'n representatie om te zetten naar iets wat ook leesbaar is voor mensen in plaats van computers. Voorbeelden zijn een uitgeschreven tekst representatie [6], een animatie of een stripverhaal.

Door een voorgaande ontwerpprojectgroep is gewerkt aan het genereren van animaties. Dit project heet Annie the Animator [7]. Het doel van het ontwerpproject dat in dit verslag wordt beschreven was om op basis van Annie een systeem te bouwen om stripafbeeldingen te genereren voor de verhalen uit de Virtual Storyteller. We hanteren hetzelfde systeem voor naamgeving als het systeem waarop we voortbouwen, daarom hebben we ons project Carrie the Cartoonist genoemd.

In dit verslag worden het ontwerpproces en de resultaten van Carrie beschreven. In hoofdstuk 2 staat het projectplan, waarin wordt beschreven wat het doel van het project is en hoe het aangepakt is. Dan volgt het programma van eisen in hoofdstuk 3. Een beschrijving van het globale ontwerp is te vinden in hoofdstuk 4. In hoofdstuk 5 wordt de implementatie van het programma toegelicht aan de hand van een uitgebreid voorbeeld, tevens wordt beschreven welke problemen we tegenkwamen bij het implementeren en hoe deze zijn opgelost. Daarna volgt een beschrijving van het testen van het programma in hoofdstuk 6. In de conclusies (hoofdstuk 7) wordt beschreven hoe ver we nu gekomen zijn met het programma, en hoe dat relateert aan het programma van eisen. Het verslag wordt afgesloten met een aantal aanbevelingen en ideeën voor verder werk in hoofdstuk 8.

Hiernaast is ook een programmahandleiding toegevoegd aan het verslag als bijlage. Hierin is voor verschillende soorten gebruikers beschreven hoe ze het programma kunnen gebruiken, configureren en uitbreiden.

Hoofdstuk 2

Projectplan

2.1 Doel

Het doel van dit project is het maken van een strip van een simpel verhaal. Hierbij ligt de focus op het maken van een herkenbare representatie van het verhaal. Realisme is hierbij van minder belang. Uiteindelijk is het de bedoeling dat de verhalen die de Virtual Storyteller [14] genereert door Carrie the Cartoonist gevisualiseerd kunnen worden.

2.2 Achtergrond informatie

Carrie the Cartoonist is gebaseerd op Annie the Animator [7]. Annie heeft als doel om van een plot een animatie te maken. Hoewel Annie al wel de basisstructuur heeft om animaties te genereren, is het systeem nog lang niet volledig genoeg om een willekeurig plot te kunnen visualiseren. Carrie wordt een aanpassing en uitbreiding van Annie die generieker is, omdat er zowel strips als animaties mee gegenereerd kunnen worden. Onze focus zal slechts liggen op de strips, omdat die makkelijker te visualiseren zijn. Het deel voor animaties zal dan ook onveranderd blijven.

2.3 Taakverdeling

Het ontwerp en de ontwikkeling van het systeem zal gebeuren in twee groepen. In het algemeen zal de ene groep zich bezighouden met de director en de andere met de presenter. Om te zorgen dat iedereen kennis heeft over het gehele systeem, wisselen ongeveer om de twee weken twee personen van groep.

2.4 Op te leveren documenten en producten

In de loop van dit project zullen we de volgende documenten en producten opleveren:

- Projectplan (bestaande uit dit hoofdstuk van het verslag),
- Programma van eisen,
- Ontwerpdocument:

  - Beschrijving van de globale architectuur,
  - Klassendiagrammen met daarin de klassen die belangrijk zijn voor het ontwerp.
- Implementatie:
  - Een gecompileerde, werkende versie van het programma,
  - Java broncode voorzien van commentaar,
  - Een API-documentatie voor het programma, gegenereerd uit de Javadoc in de broncode.
- Test documentatie:
  - Beschrijving van unittests en de resultaten daarvan,
  - Beschrijving van systeemtests en de resultaten daarvan.
- Programmahandleiding,
- Overige producten:
  - Een CD waarop alle producten en documenten in digitale vorm te vinden zijn,
  - Een poster over het project,
  - Een presentatie over het project.

Hoofdstuk 3

Programma van Eisen

3.1 Functionele eisen

Carrie the Cartoonist is gebaseerd op Annie the Animator [7]. Ons programma van eisen zal dan ook grotendeels gebaseerd zijn op hun programma van eisen.

3.1.1 Primaire functionaliteit

Deze sectie bevat de functionaliteit die noodzakelijk is voor het functioneren van het programma.

1. Het systeem moet de gebruiker de mogelijkheid bieden een TriG [15] bestand te openen, dat een fabula, plot en verwijzingen naar de ontologie (opgeslagen in OWL [11] bestanden) bevat.
2. Het systeem moet de informatie in de fabula vertalen naar een cartoonscript.
   (a) Het systeem moet een minimale set acties, doelen, gebeurtenissen, emoties en andere ontologie elementen kunnen verwerken om ten minste een voorbeeldfabula te visualiseren.
   (b) Het systeem moet missende informatie in de fabula aanvullen zonder de betekenis van het verhaal te veranderen.
   (c) Het systeem moet de verhaallijn(en) splitsen in logische kaders.
   (d) Het cartoonscript moet de staat (activiteit en emotie) van alle elementen (karakters en objecten) definiëren.
   (e) Het systeem moet om kunnen gaan met activiteiten die op hetzelfde moment op verschillende locaties plaatsvinden en deze weergeven zonder belangrijke informatie weg te gooien.
3. Het systeem moet het gegenereerde cartoonscript naar de gekozen presenter sturen.
   (a) Het systeem moet meerdere presenters ondersteunen.
   (b) Het systeem moet de gebruiker de mogelijkheid bieden een presenter te kiezen. De presenter is de module die het cartoonscript visualiseert.

4. De cartoonpresenter moet de inhoud van het cartoonscript visualiseren.
    (a) De cartoonpresenter moet afbeeldingen op een zodanige manier kiezen dat visuele elementen die te onderscheiden moeten zijn om het verhaal te begrijpen er ook anders uitzien, mits de afbeeldingendatabase dit toelaat.
    (b) De cartoonpresenter moet indien mogelijk een generiekere afbeelding kiezen als van een object geen specifieke afbeelding beschikbaar is (bv. een vrouw als roodkapje).
    (c) De cartoonpresenter moet op zijn minst alle acties, gebeurtenissen, emoties etc. die het cartoonscript van het voorbeeldverhaal bevat kunnen visualiseren.

3.1.2 Secundaire functionaliteit

De functionaliteit in deze sectie is niet noodzakelijk voor het functioneren van het programma, maar wordt wel als belangrijk beschouwd.

1. Het cartoonscript moet de setting (achtergrond etc.) van ieder kader definiëren.
2. Het systeem moet kaders toevoegen zonder de betekenis van het verhaal te veranderen indien deze de leesbaarheid van het verhaal ten goede komen.
3. De cartoonpresenter moet cartoonscriptcommando's waarvoor geen visuele representatie beschikbaar is als tekst weergeven.
4. Het systeem moet de gebruiker de mogelijkheid bieden om de visuele representatie van acties en entiteiten toe te voegen en aan te passen zonder het systeem te veranderen.

3.1.3 Optionele functionaliteit

Deze sectie bevat uitbreidingen van de functionaliteit die niet belangrijk zijn voor de basisfunctionaliteit van het systeem, maar wel leuk zijn om te hebben.

1. Het systeem kan de gegenereerde kaders opmaken als een stripboek (door deze op pagina's te groeperen).
    (a) Het systeem kan de grootte van kaders zodanig aanpassen dat de pagina er mooier uitziet, waarbij belangrijkere acties meer benadrukt worden.

3.2 Non functionele eisen

3.2.1 Kwaliteitseisen

1. Gebruik van bronnen
    (a) Tijdens de uitvoering van het programma mag het systeem 100 procent van de processorkracht gebruiken.
2. Responstijd

   (a) Tijdens de uitvoering van het programma moet het systeem binnen redelijke tijd (enkele seconden) reageren op gebruikersinvoer.

3. Betrouwbaarheid

   (a) Het systeem moet foute invoer op een redelijke manier afhandelen door terugkoppeling aan de gebruiker te geven zonder vast te lopen.

   (b) Het systeem is afhankelijk van hardware en andere software (OS, drivers, OWL-parser). Fouten in deze onderdelen kunnen wel leiden tot onverwacht gedrag van het systeem.

4. Uitbreidbaarheid en aanpasbaarheid

   (a) Het systeem moet zo ontworpen zijn dat het mogelijk is om presenters toe te voegen zonder de code van de director te veranderen.

   (b) Veranderingen in de ontologie kunnen veranderingen in de director nodig maken. Het systeem moet echter zo ontworpen worden dat er zo min mogelijk veranderingen in de bestaande code nodig zijn. Bij voorkeur is alleen uitbreiding van de code nodig.

   (c) Het systeem gebruikt Engels als uitvoertaal, maar biedt de mogelijkheid voor uitvoer in andere talen.

5. Herbruikbaarheid

   (a) De systeemcomponenten moeten op een zodanige manier ontworpen zijn dat ze zo veel mogelijk herbruikbaar zijn.

6. Bruikbaarheid

   (a) Het systeem moet eenvoudig te gebruiken zijn. Na een initiële setup, waarbij de benodigde parameters worden opgegeven, zou de gebruiker tijdens de uitvoering van het programma niet in hoeven te grijpen.

3.2.2 Platform eisen

1. Het platform dient Java versie 1.6 te kunnen draaien.

2. Het platform dient minimaal 512 MB aan intern geheugen te hebben.

3.2.3 Software eisen

1. Programmeertaal

   (a) Het systeem zal worden geschreven in Java 1.6.

2. Documentatie

   (a) De publieke API van het systeem moet worden gedocumenteerd met de Javadoc syntax [9].

   (b) De broncode moet zoveel mogelijk zelfdocumenterend zijn. Waar nodig of nuttig dient de code aangevuld te worden met betekenisvol commentaar.

Hoofdstuk 4

Ontwerp

Het ontwerp van Carrie the Cartoonist is opgesplitst in twee onderdelen, de director en de presenter. De director is verantwoordelijk voor het inlezen en analyseren van de fabula. De presenter is verantwoordelijk voor het genereren van de afbeeldingen. De communicatie tussen de director en de presenter verloopt via een Java representatie van een stripverhaal. Een stripverhaal wordt gerepresenteerd door een boomstructuur waarvan het klassendiagram hieronder is weergegeven.

4.1 Director

De basisfunctionaliteit van de director blijft gelijk aan de director van Annie: het omzetten van een plot naar een tussenformaat. Daarom zal de structuur van de director, zie figuur 4.1, grotendeels gelijk blijven aan die van Annie.

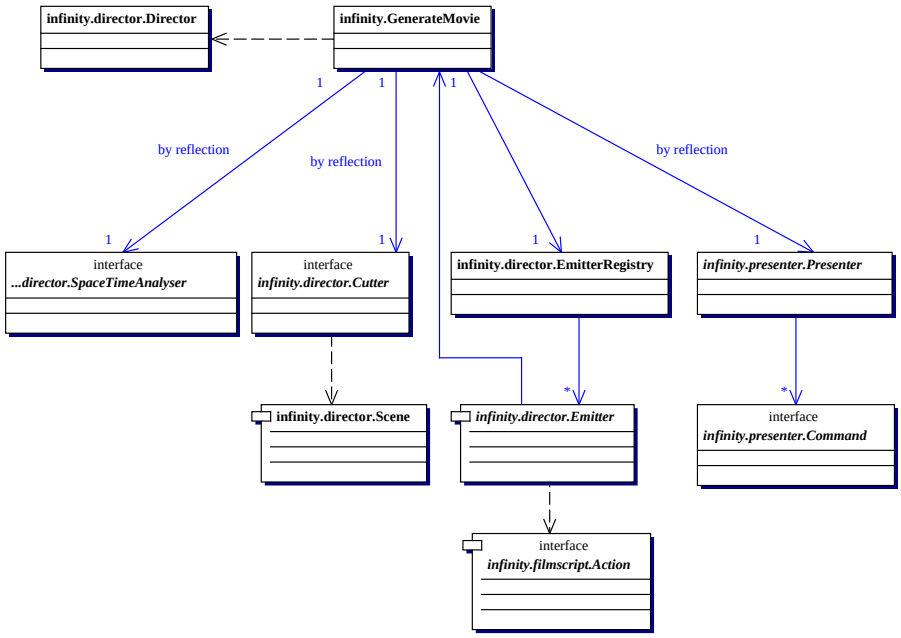
De relaties die in figuur 4.1 met 'by reflection' zijn gelabeld, worden dynamisch opgebouwd. Dit wordt gedaan zodat de instellingen uit het configuratiebestand de te laden klassen kunnen bepalen.

4.1.1 Loader

De Virtual Storyteller genereert een fabula in de vorm van een RDF document. Er bestaan verschillende notaties voor RDF documenten. De Loader van Carrie the Cartoonist is instaat om één van deze notaties, de TriG notatie, in te lezen. RDF documenten in de TriG notatie zijn compacter te schrijven dan documenten in de RDF/XML notatie. De Loader maakt gebruik van de Jena [10] API. De RDF graaf wordt met behulp van een aantal SPARQL queries ingelezen en opgeslagen in een interne Java representatie. Deze interne representatie is vervolgens de invoer voor de SpaceTimeAnalyser.

4.1.2 SpaceTimeAnalyser en Cutter

De SpaceTimeAnalyser interface levert de mogelijkheid om, gegeven de invoer van het systeem, van elke entiteit te bepalen wanneer deze entiteit aanwezig is op welke locatie. De Cutter bepaalt op basis hiervan wat interessante scènes zijn voor visualisatie. Meer informatie over de SpaceTimeAnalyser en de Cutter is te vinden in Annie the Animator [7].



Figuur 4.1: Director Klassendiagram

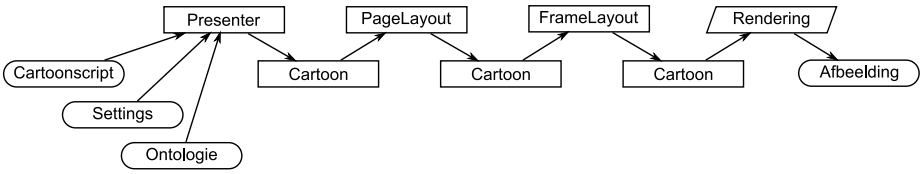
4.1.3 Emitters en Actions

De Emitters kunnen specifieke Entities uit de scènes die de Cutter genereert omzetten naar een of meerdere acties (gerepresenteerd in een Action subklasse). Om bruikbare acties te genereren, hebben we een aantal nieuwe Emitters met bijbehorende acties geschreven. Om te beginnen hebben we een AbstractActionEmitter en een AbstractAction klasse gemaakt. Deze zijn subklassen van respectievelijk Emitter en ActionBase, de basisimplementatie van de Action interface, uit Annie. De subklassen zijn zo uitgebreid, dat iedere AbstractAction een agens, patiens, instrument en doel mee kan krijgen. De AbstractActionEmitter zorgt er ook voor dat de vier voorgenoemde entiteiten gematerialiseerd worden als ze aanwezig zijn. Dit kan eventueel leiden tot dubbel gematerialiseerde entiteiten, wat verderop in het programma weer weg wordt gefilterd (zie paragraaf 4.1.4). Daarnaast zijn er Actions en ActionEmitters gemaakt voor de acties die in ons voorbeeldverhaal voorkomen. Deze kunnen deels automatisch worden gegenereerd door een door ons geschreven hulpprogramma. Na de generatie moeten bepaalde onderdelen wel nog met de hand worden ingevuld, zoals wat er met de gerelateerde entiteiten gebeurt door het plaatsvinden van de actie.

4.1.4 Commands

Klassen van het type Command worden gebruikt om de boomstructuur voor de strip op te bouwen uit de lijst met Actions. We hebben hiervoor een nieuwe set Commands geschreven die de oude Annie Command interface implementeren.

Ieder soort entiteit (bijvoorbeeld acties en locaties) heeft een eigen Command. In een configuratiebestand wordt iedere actie gekoppeld aan één commando. Een aantal Commands zijn in feite alleen een doorgeefluik; ze zetten alleen de entiteit uit de bijbe-



Figuur 4.2: Presenter proces

horende actie op de juiste plek in de cartoon boom. Sommige andere Commands (zoals het ThingCommand en ActionCommand) bouwen eerst een stukje boom op en plaatsen dat dan in de cartoon boom. Een voorbeeld: het ThingCommand maakt een referentie naar het bijbehorende ding aan en plaatst daaronder alle entiteiten die het ding draagt, vasthoudt of bevat. Daarna wordt de referentie op de juiste plek in de boom gehangen.

Daarnaast zijn er een aantal Commands die de lay-out van de strip aanbrengen in de cartoon boom. Hieronder vallen de Commands die een nieuw Frame of een nieuwe FrameGroup aanmaken.

4.1.5 Presenter koppeling

De director moet uiteindelijk een boomstructuur representatie van een stripverhaal opleveren. Deze wordt door de presenter weer als invoer gebruikt. De presenter gebruikt de informatie in de boom om de uiteindelijke strip te genereren.

4.1.6 Aanpassingen aan het oude systeem

In de director is ondersteuning voor namespaces toegevoegd. XML namespaces leveren een methode om naamconflicten te voorkomen. Met XML is mogelijk om zelf een vocabulaire van elementen te definiëren, hierdoor is het echter mogelijk dat twee personen dezelfde elementnaam gebruiken terwijl de betekenis van dit element niet noodzakelijkerwijs overeen hoeft te komen. Dit probleem wordt opgelost door een vocabulaire te kopellen aan een namespace. De volgende XML documenten zijn dus niet equivalent:

```
<?xml version="1.0"?>
<title xmlns="http://www.w3.org/1999/xhtml/">De titel</title>
```

en

```
<?xml version="1.0"?>
<title xmlns="http://www.docbook.org/ns/docbook">De titel</title>
```

Het gebruik van namespaces binnen ons project is belangrijk omdat iedereen vrij is om zelf een ontologie in OWL te specificeren. Het is dus niet uit te sluiten dat er een naamconflict optreedt indien men gebruik maakt van meerdere ontologieën.

4.2 Presenter

4.2.1 Globaal ontwerp

De presenter kant van het systeem wordt grotendeels opgebouwd als een pipeline, zie figuur 4.2. De data uit de cartoon boom wordt door een aantal opeenvolgende klassen

omgezet naar de uiteindelijke strip. Het systeem bestaat dus uit databewerkingsklassen en datarepresentatieklassen.

De pipeline begint met het laden van cartoon boom, de ontologie die de wereld beschrijft en de settings die gebruikt moeten worden. Ze worden ingeladen door hun respectievelijke Loader klassen. De data wordt omgezet naar een interne representatie die is opgeslagen in de HierarchyKnowledge en Cartoon klasse. Vervolgens wordt de PageLayout losgelaten op de data in Cartoon om te bepalen hoe de Frames uit de strip over de pagina's verdeeld moeten worden. Deze informatie wordt opgeslagen in de Page klasse. Nadat dat gebeurd is, wordt de FrameLayout aangeroepen om de kaders in de strip vorm te geven. Hierbij wordt gebruik gemaakt van de Entity klasse en haar subklassen en de klassen EntityReference en FrameLayoutConstraint. De gegenereerde informatie wordt opgeslagen in de RenderFrame klasse. De data klassen zullen methoden bevatten om de representaties van hun data te renderen.

4.2.2 Functionele eigenschappen

Een belangrijke uitdaging aan de Presenter kant van het systeem is dat twee entiteiten van hetzelfde type (bijvoorbeeld twee katten) er in de strip verschillend uit moeten zien. Hiervoor hebben we een speciaal invoerformaat gemaakt waarbij de entiteiten opgebouwd kunnen worden uit onderdelen die geselecteerd worden uit sets van mogelijkheden. Zo kunnen dus bijvoorbeeld verschillende haarstijlen, neuzen, brillen e.d. in hetzelfde bestand worden gedefinieerd. In het systeem wordt dit geregeld door bij de EntityVisualiserFactory een instantie van subklasse van EntityVisualiser op te vragen die die bepaalde entiteit kan weergeven. Die instantie maakt dan een uniek uitziende visualisatie van de entiteit.

Een tweede punt van aandacht is, dat het systeem een mechanisme moet hebben om entiteiten weer te geven als er geen direct plaatje van beschikbaar is. Net als Annie gebruiken wij hiervoor dan het meest relevante generiekere plaatje dat aanwezig is. Om te bepalen welk plaatje dan gebruikt moet worden, wordt uit de wereldontologie een hiërarchie van entiteiten gehaald die wordt opgeslagen in de klasse HierarchyKnowledge.

Ten slotte moet het systeem het mogelijk maken om emoties te laten zien. Omdat deze van frame tot frame kunnen veranderen, wilden we die informatie niet in de Entity klassen zelf opnemen. Daarom bestaat de EntityReference klasse: deze bevat een referentie naar het statische deel van de entiteit en ook de gegevens die per Frame veranderen.

4.2.3 Rendering

Voor het renderen van een strip heeft de presenter een aantal resources nodig met hierin de grafische weergave van karakters en acties, waaruit het verhaal is opgebouwd. Voor deze presenterresources is een invoerformaat nodig. Omdat het gaat om afbeeldingen, zijn er voor dit formaat twee keuzemogelijkheden: raster- en vectorafbeeldingen. Rasterafbeeldingen hebben als voordeel dat ze makkelijker te maken zijn, dit omdat een vectorafbeelding heel gemakkelijk omgezet kan worden naar een rasterafbeelding, maar niet andersom. Vectorafbeeldingen hebben als voordelen dat ze makkelijk schaalbaar zijn en dat ze makkelijk aan te passen zijn. Bij een vectorafbeelding is het bijvoorbeeld mogelijk om enkele vectoren van kleur te veranderen en zo de haarkleur te veranderen. Omdat het belangrijk is dat alle karakters er verschillend uit zien en dit

verschil in uiterlijk makkelijker te bereiken is met vectorafbeeldingen, hebben wij voor dit formaat gekozen.

Als vectorformaat is er gekozen voor SVG [13]. Dit is een open formaat dat gebaseerd is op XML [4]. Omdat XML makkelijk programmatisch te bewerken is, is dit formaat goed te gebruiken in de presenter. Daarnaast bieden veel afbeeldingbewerkingsprogramma's de mogelijkheid om afbeeldingen als SVG op te slaan, waardoor de kans groot is dat ontwerpers met hun favoriete programma de afbeeldingen kunnen maken.

Om het makkelijker te maken veel verschillende entiteiten van hetzelfde type te ondersteunen, biedt onze TemplateResource de mogelijkheid meerdere CSS [3] bestanden aan één resource te koppelen. Er is voor CSS gekozen omdat dit de gangbare standaard is om stijlinformatie aan een SVG te koppelen. Hierdoor kunnen met één resource, verschillende entiteiten worden weergegeven. Door bijvoorbeeld de keuze te bieden uit 5 verschillende haarstijlen en 3 verschillende brillen, kunnen 15 verschillende karakters worden gemaakt. Het formaat heeft de mogelijkheid dit te doen met behulp van 8 verschillende stylesheets.

Hoewel SVG afbeeldingen inmiddels door veel programma's kunnen worden weergegeven, hebben rasterafbeeldingen nog altijd een grotere ondersteuning. Daarom lijkt het ons nuttig om als uitvoer ook rasterafbeeldingen te genereren. Met de Apache Batik library [2] kunnen SVG afbeeldingen omgezet worden naar rasterafbeeldingen. Wij leveren dan ook PNG [12] afbeeldingen op in plaats van SVG afbeeldingen.

Voor het weergeven van een boek is PDF [8] het meest gebruikte formaat. Dit lijkt ons dan ook het beste formaat om te gebruiken voor het weergeven van het verhaal als stripboek. Door de SVG afbeeldingen te embedden in een XSL-FO [5] bestand kan zo'n boek gedefinieerd worden, wat met behulp van de Apache FOP library [1] om te zetten is naar PDF. Omdat XSL-FO ook een XML gebaseerd formaat is en makkelijk SVG kan embedden, lijkt dit ons een goed formaat voor het definiëren van een boek.

Voor de generatie van de uiteindelijke afbeeldingen zal als tussenuitvoer een collectie SVG afbeeldingen worden gegenereerd. Dit is omdat voor ieder karakter een eigen SVG afbeelding gemaakt dient te worden. Het zullen losse afbeeldingen moeten zijn, omdat CSS stylesheets geen scoping hebben en dus altijd op het gehele bestand van toepassing zijn.

Doordat entiteiten verschillend zijn door het toepassen van verschillende stylesheets, is het niet mogelijk deze entiteiten direct in hetzelfde bestand te plaatsen. Dit heeft tevens als gevolg dat de door Carrie gegenereerde SVG uitvoer niet makkelijk in gebruik is, omdat het uit vele bestanden bestaat. Daarom wordt SVG dan ook niet als einduitvoer opgeleverd.

4.2.4 Klassenbeschrijving

In tabel 4.1 wordt een kort overzicht gegeven van de klassen uit het ontwerp en hun verantwoordelijkheden. Zie appendix A voor het klassendiagram.

`Cartoon`	Deze klasse is verantwoordelijk voor het bijhouden van alle informatie die een strip bevat, inclusief de informatie die gegenereerd wordt door de FrameLayout en de PageLayout.
`CartoonLoader`	Deze klasse is verantwoordelijk voor het laden van een Cartoon. Deze kan bijvoorbeeld vanuit cartoonscript worden geladen, maar ook direct vanuit een datastructuur van de Director.
`Entity`	Deze klasse behoudt alle globale cartooninformatie over een bepaalde entiteit uit het verhaal (bijv. een locatie, object of karakter).
`EntityReference`	Deze klasse bevat Framespecifieke informatie over entiteiten die aanwezig zijn en refereert aan de globale Cartooninformatie.
`EntityVisualizer`	De EntityVisualizer draagt zorg voor het omzetten van de interne representatie van een entiteit naar een grafische weergave, hierbij wordt gebruik gemaakt van de informatie opgeslagen in een EntityReference.
`EntityVisualiserFactory`	Deze klasse genereert een EntityVisualizer voor een bepaalde Entity op basis van de kennis uit de HierarchyKnowledge.
`Frame`	Een Frame bevat alle informatie die nodig is om een enkel kader weer te geven.
`FrameGroup`	Een FrameGroup is een groepering van frames die niet zomaar opgesplitst kan worden.
`FrameLayout`	Deze klasse genereert de lay-out van een Frame en de RenderFrame daarvoor.
`FrameLayoutConstraint`	De FrameLayoutConstraint geeft extra layoutconstraints door aan de FrameLayout, bijvoorbeeld om twee entiteiten direct naast elkaar te zetten.
`FrameNode`	Deze klasse is onderdeel van het hiërarchiepatroon van de Frames en FrameGroups.
`Page`	Deze klasse bevat de lay-outinformatie van een pagina.
`PageLayout`	Deze klasse is verantwoordelijk voor het verdelen van de kaders over de pagina's en het genereren van de RenderPage.
`Presenter`	De Presenter is de hoofdklasse, deze is verantwoordelijk voor de 'voortstuwing' van het hele proces en het laden van alle instellingen.
`RenderContext`	De RenderContext houdt tijdens het renderen de context bij. Deze context bevat onder andere het beheer van tijdelijke bestanden.

Tabel 4.1: Verantwoordelijkheden van de klassen in de presenter

Hoofdstuk 5

Implementatie

5.1 Voorbeeld van het verloop van het programma

Om wat meer inzicht te geven in hoe het programma in elkaar zit, wordt in deze paragraaf beschreven hoe een verhaalfragment door het programma heen loopt om uiteindelijk een kader in de strip te worden. Het voorbeeldfragment is een actie van het type fk:Strike waarbij een piraat (pirate2, de agens) een andere piraat (pirate1, de patiens) aanvalt met een rapier (rapier1, het instrument). Het tijdstip waarop de actie plaatsvindt is 3000. Dit getal geeft alleen de volgorde van acties aan, niet een specifiek beginmoment vanaf tijdstip 0. Dit ziet er in TriG als volgt uit:

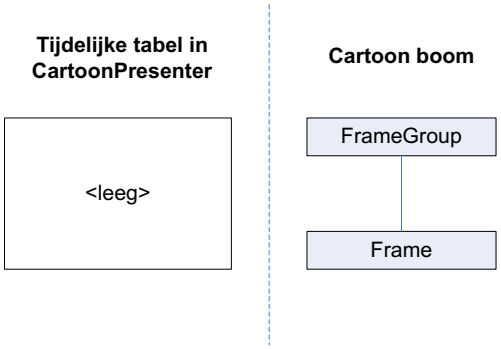
```
:action30 a fk:Strike ;
fk:agens :pirate2 ;
fk:instrument :rapier1 ;
fk:patiens :pirate1 ;
fk:time  "3000" .
```

5.1.1 Van RDF naar cartoon boom

Aangezien wij de SpaceTimeAnalyser en de Cutter niet veranderd hebben, wordt dat deel van het programma hier overgeslagen. De uitvoer van de Cutter is een lijst met Scenes, waarbij een Scene in feite een tabel is van tijdsintervallen met ieder een lijst met de entiteiten (inclusief acties) die in dat in dat tijdsinterval aanwezig zijn. Voor het bovenstaande fragment wordt maar één Scene gegenereerd, met daarin één tijdsinterval. De bijbehorende tijd is 3000 en de lijst met entiteiten bevat de Strike actie, de rapier en de twee piraten.

Nu moeten de Scenes worden omgezet naar een lijst met acties, die vervolgens worden omgezet naar een cartoon boom. ‘Actie’ betekent hier een data eenheid waarvoor of waarmee het programma een stukje cartoonboom maakt, dus niet de acties die plaatsvinden in de cartoon (zoals de Strike actie). Er zijn bijvoorbeeld acties die modelleren dat het programma een nieuw Frame aan moet maken en acties die modelleren dat een karakter een bepaalde emotie moet krijgen.

Voor iedere Scene uit de Cutter uitvoer wordt eerst een nieuwe EnterSceneAction aangemaakt. Deze actie wordt daarna direct uitgevoerd door het bijbehorende FrameGroupCommand, die een nieuwe FrameGroup in de cartoon boom zet. Voor ieder tijdsinterval binnen de Scene wordt een nieuwe NextTimeSlotAction aangemaakt.



Figuur 5.1: Programmastaat in fase één

Entiteit	**Emitter**	**Action(s)**	**Command**
Strike actie	StrikeActionEmitter	StrikeAction	ActionCommand
		3× ThingAction voor pirate1, pirate2 en rapier1	3× ThingCommand
rapier1	ThingActionEmitter	ThingAction	ThingCommand
pirate1	ThingActionEmitter	ThingAction	ThingCommand
pirate2	ThingActionEmitter	ThingAction	ThingCommand

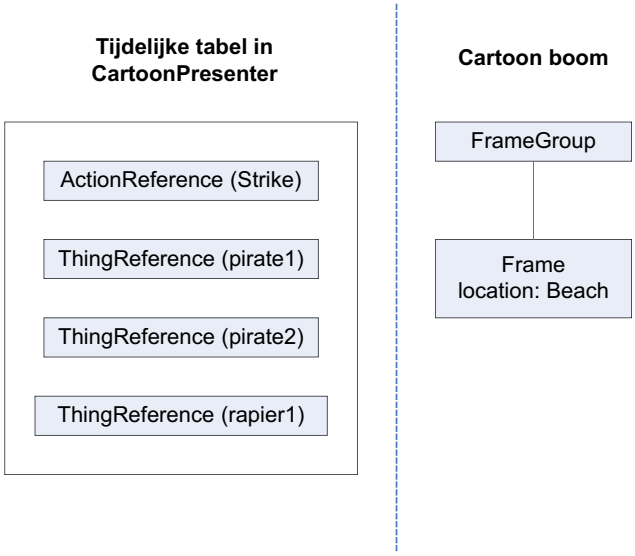
Tabel 5.1: Overzicht van koppeling tussen Emitters, Actions en Commands

Ook deze worden meteen uitgevoerd door het bijbehorende NewFrameCommand, die een nieuw Frame binnen de huidige FrameGroup aanmaakt. De staat van het programma op dit moment is schematisch weergegeven in afbeelding 5.1. Wat de tijdelijke tabel in de CartoonPresenter is, wordt verderop besproken. De actie namen zijn afkomstig uit Annie the Animator, vandaar dat de namen op film onderdelen gebaseerd zijn. De Commands zijn voor Carrie the Cartoonist opnieuw geschreven, waardoor de namen ervan wel betrekking hebben op striponderdelen.

Vervolgens wordt voor iedere entiteit en actie uit de lijst die bij het huidige tijdsinterval hoort, een geschikte Emitter uitgezocht. Een Emitter genereert voor die entiteit een of meerdere acties. Deze worden daarna door de bijbehorende Commands uitgevoerd, waardoor ze op de juiste plek in de cartoon boom gestopt worden. Zie tabel 5.1 voor een beknopt overzicht van deze koppelingen.

De Emitter voor de Strike actie genereert ook ThingActions voor zijn agens, patiens en instrument. In dit geval levert dat dubbele acties op, maar helaas is de SpaceTimeAnalyser van Annie niet helemaal consequent in het materialiseren van actie onderdelen, waardoor soms onderdelen niet gematerialiseerd zouden worden als we het niet op deze manier oplossen. De oorzaak lijkt te zijn dat de SpaceTimeAnalyser in de war raakt van bewegingen over TransitWays. Meer hierover is te vinden in paragraaf 5.2.4. De Rapier, Piraat 1 en Piraat 2 worden alle drie ook nog een keer door een eigen ThingActionEmitter omgezet naar een ThingAction.

Nu wordt het ActionCommand losgelaten op de StrikeAction. Er wordt een ActionReference aangemaakt voor de StrikeAction. Deze wordt gevuld met de informatie uit StrikeAction; dat wil zeggen dat de agens, patiens en het instrument worden gezet. De ActionReference wordt nog niet meteen in het huidige Frame gezet, maar tijdelijk



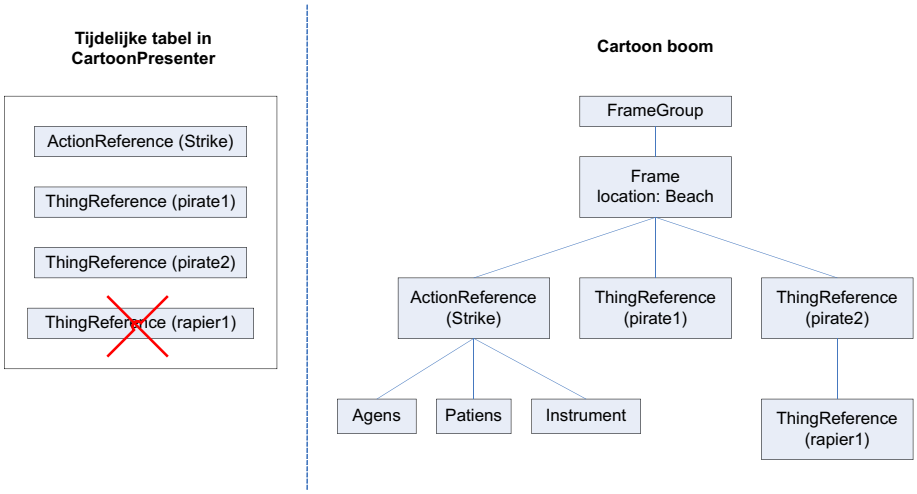
Figuur 5.2: Programmastaat in fase twee

bewaard in een tabel in de CartoonPresenter. De agens, patiens en instrument worden hier ook meteen opgeslagen als ThingReference.

Vervolgens worden de ThingActions van pirate1, pirate2 en rapier omgezet door ThingCommand objecten. Het ThingCommand doet twee dingen: eerst probeert het de locatie waar het frame zich afspeelt vast te stellen aan de hand van het Thing, als dat nog niet gebeurd is. Ten tweede wordt van het Thing een ThingReference gemaakt, die ook tijdelijk wordt bewaard in een tabel in CartoonPresenter. In dit geval heeft de rapier geen 'locatedAt' eigenschap, maar de twee piraten wel. De eerste piraat die door een Command bekeken wordt, bepaalt dan ook de locatie van het Frame. Aangezien een Frame maar één locatie heeft, maakt het niet uit welke entiteit deze bepaalt, want in een consistent verhaal hebben alle entiteiten die bij een actie betrokken zijn dezelfde locatie. Beide piraten en de rapier worden omgezet naar ThingReferences. In dit geval bestonden er al referenties naar deze dingen omdat ze bij de StrikeAction ook al gemaakt worden. Deze referenties worden nu recursief verder ingevuld. Een voorbeeld: pirate2 wordt omgezet naar een referentie. Alle dingen die hij bij zich draagt, in zijn hand heeft of bevat worden in de referentie gestopt, en alles wat die dingen weer bij zich hebben wordt weer onder de referentie van dat ding gestopt, enzovoort. In dit geval houdt het opbouwen na één iteratie op, want pirate2 heeft rapier1 vast, maar rapier1 heeft zelf niets meer vast. De staat van het programma op dit moment is te zien in afbeelding 5.2. In de figuur is de inhoud van alle References (zoals agens en instrument bij de actie) niet weergegeven.

Er is echter nog een volledige set ThingActions, die ook door ThingCommands worden omgezet. Deze Commands voegen echter niets meer toe aan de map met referenties in CartoonPresenter, omdat alle dingen al eerder gematerialiseerd en ingevuld zijn. De staat van het programma blijft hetzelfde.

Nu is er in CartoonPresenter dus een map met alle EntityReferences die potentieel op het huidige frame moeten komen. In dit voorbeeld zijn dat referenties voor de StrikeAction, pirate1, pirate2, en rapier1. Om het maken van het Frame af te sluiten, wordt



Figuur 5.3: Programmastaat in fase drie

nog een EndOfTimeSlotAction gegenereerd met bijbehorende EndOfFrameCommand. Deze zorgen er samen voor dat alleen de referenties die ook echt op zichzelf staan in het Frame terecht komen. In dit stadium wordt rapier1 uit de map gefilterd, omdat die niet op zichzelf staat, maar wordt gedragen door pirate2. Alle andere referenties worden wel toegevoegd aan het Frame. De nieuwe staat van het programma wordt weergegeven in afbeelding 5.3. Hierna worden nogmaals alle acties langsgelopen om hun postExecute methode aan te roepen. Hiermee kunnen acties dingen aan de wereld veranderen nadat de actie heeft plaatsgevonden. De StrikeAction uit dit voorbeeld maakt daar geen gebruik van, maar bijvoorbeeld een WalkAction zou na het lopen de agens verplaatsen naar de nieuwe locatie.

Het cartoon boom fragment dat nu is opgebouwd, is te zien in figuur 5.4 en is enigszins ingekort voor de leesbaarheid. In paragraaf 'Cartoon boom Afdrukken' van de handleiding wordt beschreven hoe deze boom gegenereerd kan worden.

Het Relevance veld kan door de Cutter gezet worden om aan te geven hoe belangrijk dit deel van de cartoonboom is. Hier kan bij de lay-out rekening mee worden gehouden door belangrijke dingen prominenter in beeld te brengen. Vooralsnog is de waarde van Relevance altijd 1.0.

Het StateClass argument bij ThingReferences kan worden gebruikt om gegevens als de emotie van de entiteit (meestal karakter) op te slaan. In de volgende sectie wordt beschreven hoe dit stuk cartoonboom wordt omgezet naar de uiteindelijke grafische weergave.

5.1.2 Van cartoon boom naar strip

De presenter voert de volgende stappen uit:

1. Cartoon laden
2. Visualizers van alle entiteiten laden
3. Pagina's layouten

```
Cartoon:
    FrameGroup (Relevance=1.0):
        Frame (Relevance=1.0):
            @location:
                EntityReference:
                    Location ([#Beach]):
            ThingReference (StateClass=null):
                Item ([#FemalePirate])
            ActionReference: [#Strike]
                Agens:
                ThingReference (StateClass=null):
                    Item ([#MalePirate])
                    Holding:
                        ThingReference (StateClass=null):
                            Item ([#Rapier])
                Patiens:
                ThingReference (StateClass=null):
                    Item ([#FemalePirate])
                Instrument:
                ThingReference (StateClass=null):
                    Item ([#Rapier])
                Target:
                <niet aanwezig>
            ThingReference (StateClass=null):
                Item ([#MalePirate])
                Holding:
                    ThingReference (StateClass=null):
                        Item ([#Rapier])
```

Figuur 5.4: Weergave van de cartoon boom

4. Kaders layouten

5. Afbeelding renderen

Cartoon laden Eerst wordt de cartoonboom geladen door de CartoonLoader. Momenteel is alleen de CartoonPresenter een implementatie van een CartoonLoader. De director genereert op deze manier direct de cartoonboom die nodig is voor de presenter, waardoor er geen tussenstap is tussen de uitvoer van de director en de invoer van de presenter.

Visualizers van alle entiteiten laden Voor alle entiteiten uit de cartoon wordt een Visualiser geladen, die deze entiteiten in de gehele strip zal visualiseren.

Pagina's layouten Hierna wordt de pagina door de PageLayout opgemaakt, die kan beslissen welke kaders op welke pagina komen en hoe groot ze daar zijn. Momenteel bestaat alleen de SimplePageLayout die één kader per pagina lay-out, met allemaal gelijke grootte.

Kaders layouten Nu worden alle kaders opgemaakt. Hierbij wordt bepaald waar alle entiteiten in het kader staan. De SimpleSVGFrameLayout plaats alle entiteiten van links naar rechts. De SVGFrameLayout vraagt aan alle Visualizers de FrameLayout-Constraints op en gebruikt deze om te zorgen dat entiteiten indien gewenst naast elkaar komen te staan of aan de rand van de afbeelding komen te staan.

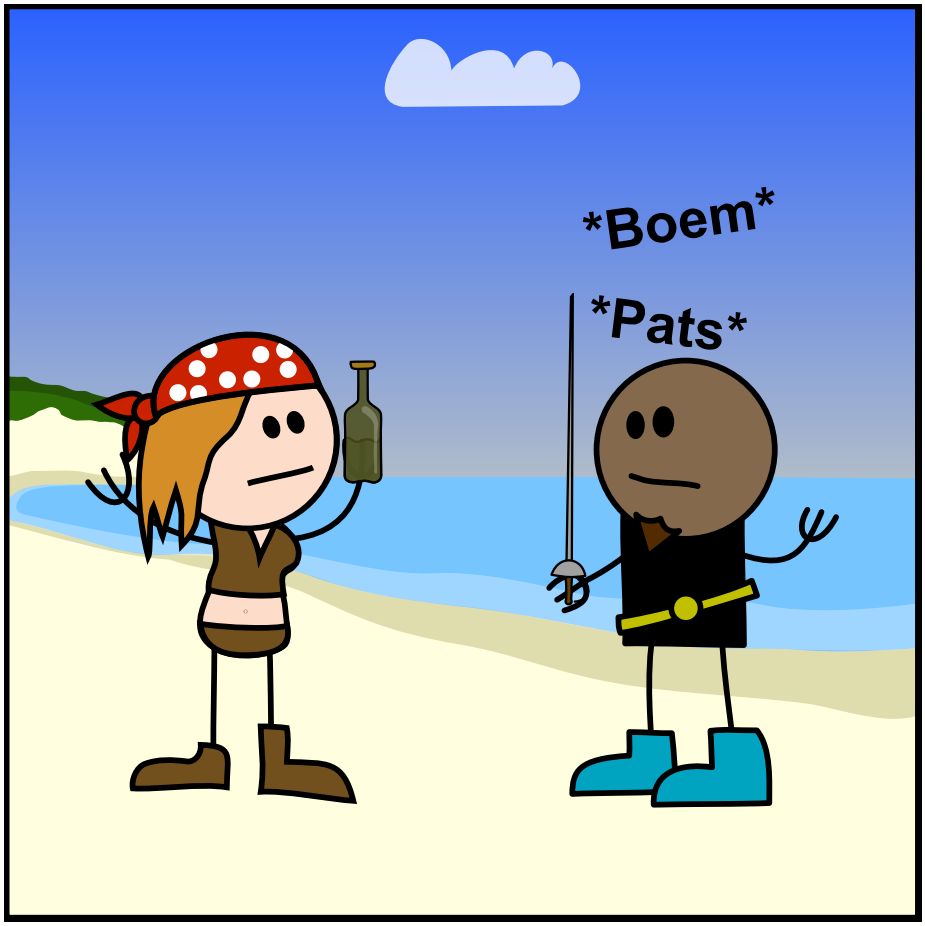
Afbeelding renderen Nu alle kaders zijn opgemaakt, kan de cartoon naar een afbeelding worden gerenderd. In deze stap wordt de SVGRenderCartoon gebruikt om de strip met behulp van SVG te renderen. Hierbij wordt eerst de strip recursief naar een hele set van tijdelijke bestanden gerenderd, waarna deze worden omgezet naar PNG en PDF, zie afbeelding 5.5 voor het resultaat. Dit is de uiteindelijke uitvoer van de presenter.

5.2 Problemen bij de implementatie

In deze paragraaf worden een aantal problemen beschreven die we tijdens het maken van Carrie the Cartoonist tegen zijn gekomen.

5.2.1 Gebruik maken van eerder gedaan werk

Inmiddels is wel duidelijk dat Carrie the Cartoonist voor een groot deel gebaseerd is op Annie the Animator [7]. Een van de grotere problemen waar we tegenaan zijn gelopen bij dit project is, dat het erg moeilijk in te schatten is hoeveel tijd er nodig is om je in te werken in andermans code. Hierdoor bleek onze oorspronkelijke planning niet geheel haalbaar te zijn, en hebben we al onze mijlpalen een plek door moeten schuiven, waarbij de laatste mijlpaal (met optionele functionaliteit) van de planning af is gevallen.



Figuur 5.5: Gerenderd voorbeeld uit hoofdstuk 5.1

Manier	Notatie
Volledig	< URI >
Afgekort	prefix : localname

Tabel 5.2: Notatie van URI's in TriG

5.2.2 Verkeerd gebruik van XML & RDF

Carrie the Cartoonist bouwt voort op het werk van Annie the Animator. Omdat de Virtual Storyteller (nog) geen RDF uitvoer oplevert, heeft de vorige groep zelf een voorbeeldfabula in RDF opgesteld. Deze voorbeeldfabula is geschreven in de TriG notatie. Maar de geschreven voorbeeldfabula bevat echter een tweetal problemen. We laten een deel van de oude fabula zien om dit toe te lichten:

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix swc:     <http://www.owl-ontologies.com/
                  StoryWorldCore.owl#> .
@prefix fk:      <http://www.owl-ontologies.com/
                  FabulaKnowledge.owl#> .

<http://www.owl-ontologies.com/Graphs.owl#graph_1> {
    <http://www.owl-ontologies.com/FabulaKnowledge.owl#Grandma>
          a       <fk:Character> ;
          rdfs:label "unlabeled" ;
          <swc:locatedAt> <http://www.owl-ontologies.com/
                           FabulaKnowledge.owl#House2> .
}
```

In de TriG notatie kunnen URI's om twee manieren worden genoteerd: (1) volledig uitgeschreven of (2) afgekort m.b.v. een prefix. De notatie van beide staat in de tabel 5.2. Opgemerkt moet worden dat a een afkorting voor <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> is.

In het bovenstaande stukje fabula is op twee plaatsen een afgekorte URI referentie alsnog omgeven door < en > symbolen, namelijk bij <fk:Character> en <swc:locatedAt>. We hebben dit opgelost door de overbodige <> en symbolen te verwijderen. Hierdoor verandert de betekenis van de fabula. De fabula werd hierdoor correct, maar de programmacode welke verantwoordelijk was voor het inlezen van het TriG bestand bleek hierna niet meer te functioneren. Uiteindelijk is besloten om dit gedeelte te herschrijven.

5.2.3 Betekenis hechten aan de subject identifier

Het tweede probleem kwam voort uit het feit dat het oude systeem de subject identifier gebruikte om een koppeling met de ontologie tot stand te brengen. De waarde van de subject identifiers kan men echter veranderen zonder dat de RDF graaf veranderd. RDF software zoals reasoners gebruiken dan ook niet deze informatie. Wij hebben besloten om de koppeling met de ontologie tot stand te brengen m.b.v. het a predicaat.

Ter vergelijking met het vorige voorbeeld, dat dit niet op de juiste wijze deed, geven we een stukje uit de vernieuwde fabula:

```
[... prefixes uit vorig voorbeeld ..]
```

```
@prefix :        <http://tempest.student.utwente.nl/~carrie/
                  fabulae/roodkapje-1\#> .

:graph_1 {
    :Grandma
          a       fk:Character ;
          rdfs:label "unlabeled" ;
          swc:locatedAt fk:House2 .
```

In dit voorbeeld bestaat er een entiteit genaamd Grandma van het type fk:Character. In het oude systeem werden afbeeldingen op basis van deze naam gekoppeld aan de instantie. De naam heeft echter geen betekenis, dus de koppeling zou op basis van het type moeten gebeuren. In het gegeven voorbeeld geeft het type echter weinig informatie, omdat het fk:Character is. Het ligt dus voor de hand een nieuw type Grandma te definiëren om een specifiekere afbeelding te kunnen gebruiken.

5.2.4 BaseSpaceTimeAnalyser

De BaseSpaceTimeAnalyser heeft een aantal problemen. Het doel van een SpaceTimeAnalyser is om een lijst op de leveren van wat wanneer waar is of gebeurt. Hierbij treden een aantal problemen op, waardoor de gegenereerde lijst niet overeenkomt met de werkelijkheid. Deze problemen worden hier beschreven en een mogelijke oplossing kan gevonden worden in hoofdstuk 8.1.

Dubbelop implementeren van acties

De huidige aanpak heeft als gevolg dat alle acties nu twee maal geïmplemeteerd moeten worden: één maal in de Virtual Storyteller en één maal in ons systeem. Dit is het gevolg van de opbouw van het tussenformaat. Deze bevat namelijk de beginsituatie van de wereld en vervolgens wordt aangegeven welke acties er worden uitgevoerd. Wat deze acties tot gevolg hebben wordt niet in de fabula opgenomen. Een voorbeeld:

```
# Begin situatie
:main {
    :rapier a ps:Rapier ;
        rdfs:label "Een soort zwaard!" .

    :pirate a cs:Pirate ;
        swc:holds :rapier .

    :victim a cs:Pirate .

    :strike a fk:Strike ;
        fk:agens :pirate ;
        fk:instrument :rapier ;
        fk:patiens :victim ;
        fk:time  "1" .
}
```

In de Virtual Storyteller is geprogrammeerd wat een Strike actie tot gevolg heeft. De staat van de wereld zal hierop aangepast worden, het slachtoffer kan b.v. gewond

```
# Piraat 1 vindt het leuk om Piraat 2 aan te vallen
:emotion1 a fk:Joy;
fk:character :pirate1 ;
fk:time "3002" .
```

Figuur 5.6: Voorbeeld van een emotie

raken. Dit staat echter niet in de fabula vermeld. Wat de gevolgen zijn van iedere actie zal dus nogmaals geïmplementeerd moeten worden in ons systeem. Het spreekt voor zich dat dit geen efficiënte manier van werken is.

In de BaseSpaceTimeAnalyser zijn echter lang niet alle acties geïmplementeerd, met als gevolg dat de uitvoer niet de juiste wereldstaat weergeeft.

Bepalen van locaties

De Walk actie is wel geïmplementeerd in de BaseSpaceTimeAnalyser, maar verandert niet de daadwerkelijke RDF. Het gevolg hiervan is dat als een locatie verandert, in de RDF nog de oude locatie staat vermeld. Hierdoor zal de verandering van de locatie slechts op het tijdstip van de verandering bekend zijn. Op een later tijdstip is de locatie weer gelijk aan de originele locatie.

Verkeerd gebruik van relaties

Alle relaties die een fk:Character bevatten, worden gebruikt om de locatie van een karakter te bepalen. Dit gaat mis in de situatie waarin een emotie wordt gebruikt, omdat deze geen locatie maar wel een fk:Character heeft, zie figuur 5.6. Hierdoor wordt dus ook de locatie van het karakter waar naar verwezen wordt verwijderd, omdat deze gelijk dient te zijn aan die van de emotie.

5.2.5 Afwezigheid van documentatie over RDF formaat en voorbeeldfabula

Tijdens de verbetering van de deserializatie van de RDF fabula bleek dat het onduidelijk was welk formaat de graaf precies had. Helaas was hierover nog geen documentatie beschikbaar, dit zorgde voor vertraging van de implementatie. Ivo Swartjes heeft in reactie hierop een korte uitleg geschreven over het te hanteren formaat. Door de wisseling van verhaaldomein tijdens het project was er geen voorbeeldfabula, deze is door de projectgroep zelf gemaakt. Hierdoor bevat de voorbeeldfabula waarschijnlijk niet alle vereiste velden. Het zou handig zijn als er een referentie fabula is, die een volledig beeld geeft van alle verplichte en mogelijke fabulaconstructies (denk hierbij aan subgrafen met context en emoties van karakters).

Hoofdstuk 6

Testen

In dit hoofdstuk wordt beschreven hoe getest is of Carrie werkt zoals de bedoeling is.

6.1 Unit tests

De director kant van Carrie is weinig veranderd ten opzichte van de director in Annie the Animator. Daarom konden de unit tests die voor de Annie director gemaakt zijn, hergebruikt worden. Hiermee wordt nog steeds alle belangrijke functionaliteit aan de director kant getest. Het draaien van de unittests in Eclipse wijst uit dat het programma nog steeds alle tests doorstaat.

6.2 Systeem tests

Testen van het hele systeem wordt gedaan door een testfabula, genaamd `pirates-1.trig`, in te voeren en te bekijken of de uitvoer er uitziet zoals we verwachten. We hebben het testverhaal tijdens het schrijven van het programma veelvuldig gebruikt om het programma als geheel te testen. Verder is er nog een tweede testverhaal, `pirates-2.trig`, dat bestaat uit één tijdstap met veel karakters en acties. Het eerste testverhaal lopen we stap voor stap door om te kijken of iedere afbeelding overeenkomt met onze verwachtingen.

6.2.1 Beginsituatie

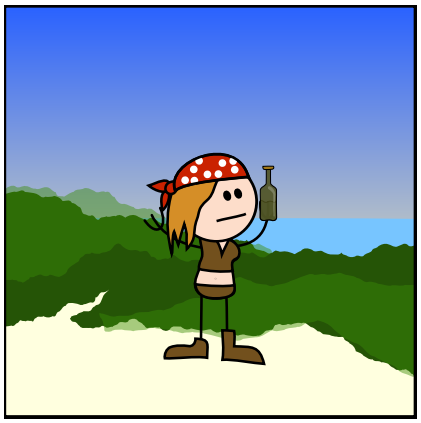
We bekijken eerst de beginsituatie van de wereld. De relevante delen uit de fabula zijn:

```
# De caribische zee
:sea1 a ps:Sea ;
    swc:hasRegion :archipelago1 .

# De 'Black stone' archipelago
:archipelago1 a swc:GeographicArea ;
    swc:hasRegion :island1 ;
    rdfs:label "Black stone archipelago" .

# Blood cliff isle
:island1 a ps:Island ;
    rdfs:label "Blood Isle" .
:island1_inland a ps:Inland ; swc:partOfGeographicArea :island1 .
:island1_beach1 a ps:Beach ; swc:partOfGeographicArea :island1 .
:island1_beach2 a ps:Beach ; swc:partOfGeographicArea :island1 .
```

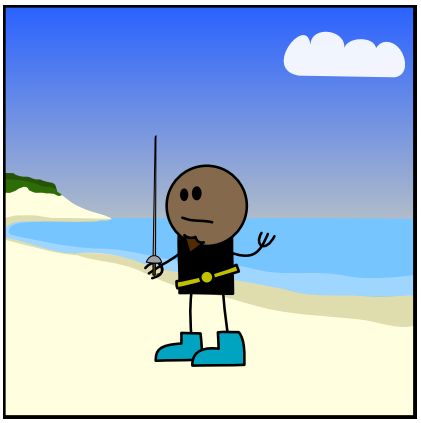
```
#verbind  inland en beach1
:beach_path_a a swc:GroundWay ;
    swc:fromGeographicArea :island1_beach1 ;
    swc:toGeographicArea :island1_inland .
:beach_path_b a swc:GroundWay ;
    swc:fromGeographicArea :island1_inland ;
    swc:toGeographicArea :island1_beach1 .

#verbind beach1 en beach2
:beach_connector_a a swc:GroundWay ;
    swc:fromGeographicArea :island1_beach1 ;
    swc:toGeographicArea :island1_beach2 .
:beach_connector_b a swc:GroundWay ;
    swc:fromGeographicArea :island1_beach2 ;
    swc:toGeographicArea :island1_beach1 .

#setup (cast en items)

#Rum
:rum a ps:Rum ;
    rdfs:label "Rum iemand?" ;
    swc:containedBy :bottle_o_rum . # contains
:bottle_o_rum a ps:Bottle ;
    rdfs:label "Een fles." ;
    swc:contains :rum .

#Rapier
:rapier1 a ps:Rapier ;
    rdfs:label "Een soort zwaard!" .

#Cast
:pirate1 a cs:FemalePirate ;
    swc:holds :bottle_o_rum ; # holds
    swc:locatedAt :island1_inland .

:pirate2 a cs:MalePirate ;
    swc:holds :rapier1 ; # holds
    swc:locatedAt :island1_beach2 .
```

Hierin wordt de volgende situatie omschreven:

- `sea1` is een zee en heeft een regio genaamd `archipelago1`,
- `archipelago1` is een geografisch gebied en heeft een regio genaamd `island1`,
- `island1` is een eiland,
- `island1_inland` is een stukje binnenland dat onderdeel is van `island1`,
- `island1_beach1` is een strand dat onderdeel is van `island1`,
- `island1_beach2` is een strand dat onderdeel is van `island2`,
- `beach_path_a` is een pad dat `island1_inland` verbindt met `island_beach1`,
- `beach_path_b` is een pad dat `island1_beach1` verbindt met `island1_inland`,
- `beach_connector_a` is een pad dat `island_beach1` verbindt met `island_beach2`,
- `beach_connector_b` is een pad dat `island_beach2` verbindt met `island_beach1`,
- `rum` is rum die zich bevindt in `bottle_o_rum`,
- `bottle_o_rum` is een fles waarin zich `rum` bevindt,
- `rapier1` is een rapier, dit is een soort zwaard,



Figuur 6.1: Eerste kader uit de testfabula



Figuur 6.2: Tweede kader van de testfabula

- `pirate1` is een vrouwelijke piraat die zich bevindt op `island_inland` en die `bottle_o_rum` vasthoudt en
- `pirate2` is een manlijke piraat die zich bevindt op `island_beach2` en die `rapier1` vasthoudt.

6.2.2 Eerste kader

De eerste actie in de fabula is een loop actie. Hierin loopt `pirate1` naar `island1_beach`. Dit is hieronder weergegeven in TriG notatie:

```
:action10 a fk:Walk ;
    fk:agens :pirate1 ;
    fk:instrument :beach_path_b ;
    fk:target :island1_beach1 ;
    fk:time  "1000" .
```

Het resultaat van deze actie is te zien in afbeelding 6.1. In het kader zien we:

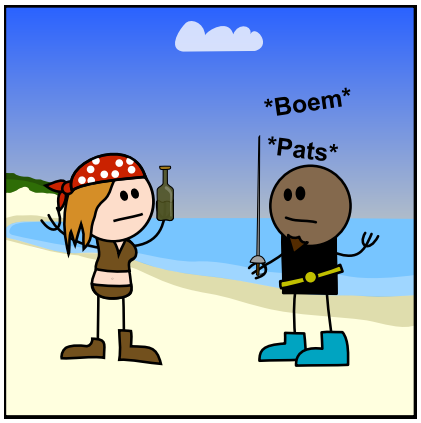
- Een vrouwelijke piraat, dit volgt uit de actie,
- Een gevulde fles rum in de hand, dit volgt uit de beginsituatie en
- De piraat bevindt zich in het binnenland, dit volgt uit de beginsituatie.

Het gevolg van deze actie is dat de vrouwelijke piraat zich nu op strand 1 bevindt.

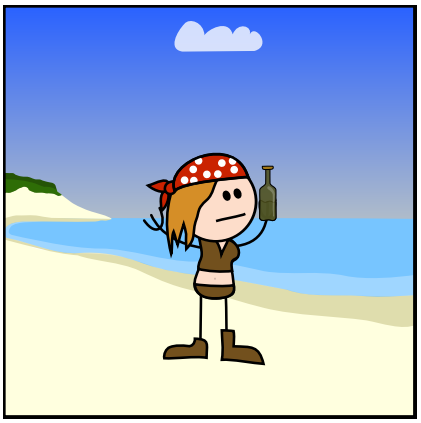
6.2.3 Tweede kader

In het tweede kader loopt een mannelijke piraat naar hetzelfde strand. De volgende RDF is hiervoor verantwoordelijk:

```
:action20 a fk:Walk ;
    fk:agens :pirate2 ;
    fk:instrument :beach_connector_b ;
    fk:target :island1_beach1 ;
    fk:time  "2000" .
```



Figuur 6.3: Derde kader van de testfabula



Figuur 6.4: Vierde kader van de testfabula

In afbeelding 6.2 zien we het volgende:

- Een mannelijke piraat, dit volgt uit de actie,
- Een rapier in de hand van de piraat, dit volgt uit de beginsituatie en
- De piraat bevindt zich op strand 2, dit volgt uit de beginsituatie.

Het gevolg van deze actie is dat de mannelijke piraat zich nu ook op strand 1 bevindt.

6.2.4 Derde kader

In het derde kader wordt het verhaal pas echt interessant. De mannelijke piraat valt de vrouwelijke piraat aan. Dit volgt uit de volgende RDF:

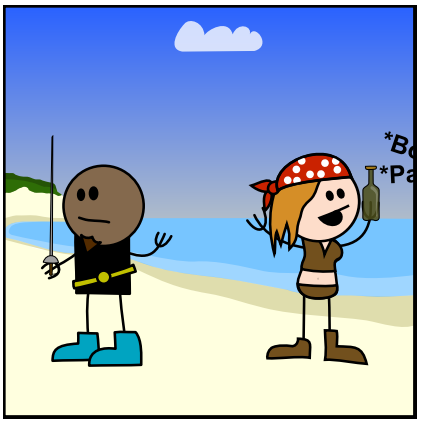
```
:action30 a fk:Strike ;
    fk:agens :pirate2 ;
    fk:instrument :rapier1 ;
    fk:patiens :pirate1 ;
    fk:time  "3000" .
```

In het resulterende kader, afbeelding 6.3, zien we het volgende:

- Een vrouwelijke piraat op strand 1, dit volgt uit de eerste actie,
- Een mannelijke piraat op strand 1, dit volgt uit de twee actie,
- De vrouwelijke piraat houdt de fles met rum vast, dit volgt uit de beginsituatie,
- De mannelijke piraat houdt de rapier vast, dit volgt uit de beginsituatie,
- De mannelijke piraat valt de vrouwelijke piraat aan, dit volgt uit de actie en
- Het verschil tussen strand 1 en strand 2 blijkt uit de verschillende wolkjes.

Deze actie blijft zonder gevolgen.

Figuur 6.5: Vijfde kader van de testfabula



Figuur 6.6: Zesde kader van de testfabula

6.2.5 Vierde kader

Het vierde kader, te zien in afbeelding 6.4, wordt veroorzaakt door een fout in de SpaceTimeAnalyser. Doordat de SpaceTimeAnalyser problemen heeft met het bepalen van locaties, zoals bescheven in paragraaf 5.2.4, wordt een actie door de Cutter over meerdere kaders verdeeld. De entiteiten die betrokken zijn bij de actie waarvoor dit kader wordt gegenereerd, zijn immers op verschillende locaties. Dit is tevens de reden dat niet beide piraten aanwezig zijn in dit kader. In het kader waar de actie daadwerkelijk gebeurt, hier het derde kader, zijn wel alle entiteiten aanwezig omdat de ActionEmitters zorgen dat alle betrokkenen aanwezig zijn.

6.2.6 Vijfde kader

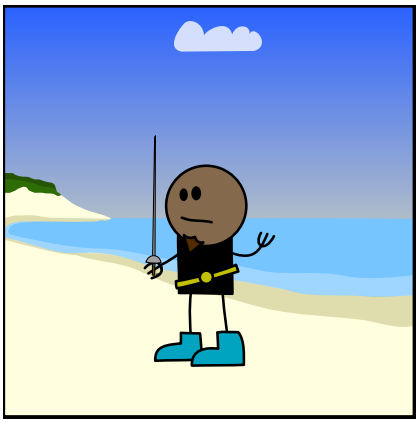
De vrouwelijke piraat drinkt in het vijfde kader, afgebeeld in afbeelding 6.5, de fles rum leeg. Dit is het gevolg van de volgende RDF:

```
:action31 a fk:Drink ;
    fk:agens :pirate1 ;
    fk:patiens :bottle_o_rum ;
    fk:target :rum ;
    fk:time "3001" .
```

We bekijken wederom het resultaat:

- Een vrouwelijke piraat op strand 1, dit volgt uit de eerste actie,
- Er is geen mannelijke piraat op strand 1. Deze zou echter wel aanwezig moeten zijn, zoals blijkt uit de gevolgen van actie 2,
- De vrouwelijke piraat heeft een fles rum, dit volgt uit de beginsituatie en
- De vrouwelijke piraat drinkt de fles rum leeg. Dit blijkt uit de actie.

Als gevolg van deze actie is de fles rum leeg gedronken.



Figuur 6.7: Zevende kader van de testfabula

6.2.7 Zesde kader

In afbeelding 6.6 neemt de vrouwelijke piraat wraak en slaat terug met de lege fles. De vrouwelijke piraat vindt het leuk om deze actie uit te voeren. Dit volgt uit de volgende RDF:

```
:action32 a fk:Strike ;
    fk:agens :pirate1 ;
    fk:instrument :bottle_o_rum ;
    fk:patiens :pirate2 ;
    fk:time  "3002" .

:emotion1 a fk:Joy;
    fk:character :pirate1 ;
    fk:time "3002" .
```

- Een vrouwelijke piraat op strand 1, dit volgt uit de eerste actie,
- Een mannelijke piraat op strand 1, dit volgt uit de twee actie,
- De vrouwelijke piraat heeft een lege fles rum, dit volgt uit het vijfde kader,
- De mannelijke piraat heeft een rapier,
- De vrouwelijke piraat vindt het leuk op terug te slaan, dit blijkt uit het lachende gezicht.

Deze actie blijft zonder gevolgen.

6.2.8 Zevende kader

Het zevende kader, ingevoegd als afbeelding 6.7, wordt wederom veroorzaakt door de problemen met de SpaceTimeAnalyser, zoals bescheven bij het vierde kader.

Hoofdstuk 7

Conclusies

In dit hoofdstuk wordt beschreven hoe ver we precies zijn gekomen met het project. Ook wordt teruggegrepen naar het programma van eisen: in hoeverre voldoet het gemaakte product aan de eisen die waren gesteld?

7.1 Stand van zaken

Het basisprogramma voor het omzetten van een Virtual Storyteller verhaal naar een stripverhaal is helemaal af. Op dit moment bestaan er afbeeldingen voor een beperkte subset van het domein dat momenteel gebruikt wordt: piraten. Voor andere verhaaldomeinen kan wel een strip gegenereerd worden, maar in zo'n strip zal veel gebruik worden gemaakt van de standaard Visualizer, oftewel: een rechthoek met de naam van de entiteit erin. Om voor andere domeinen mooie uitvoer te genereren, moeten er nog veel afbeeldingen worden toegevoegd aan het systeem.

7.2 Het programma versus De eisen

In deze paragraaf wordt meer in detail ingegaan op wat het basisprogramma wel en niet kan, gerelateerd aan het programma van eisen zoals beschreven in hoofdstuk 3. In het kort komt het erop neer dat het programma aan vrijwel alle eisen voldoet. Sommige eisen bleken achteraf niet van toepassing te zijn, een enkele secundaire of optionele eis is nog niet geïmplementeerd.

7.2.1 Functionele eisen

De functionele eisen zijn opgedeeld in primaire, secundaire en optionele eisen. Ze worden hieronder per soort besproken.

Primaire functionaliteit

Het programma voldoet aan de meeste primaire functionele eisen (te vinden in paragraaf 3.1.1), zoals de mogelijkheid tot het laden van een TriG bestand (eis 1a), het visualiseren van minstens één voorbeeldverhaal (eis 2a), verschillende karakters een ander uiterlijk geven (eis 4a) en het terugvallen op een generiekere afbeelding als de specifieke afbeelding niet aanwezig is (eis 4b).

Het programma vult op een aantal punten missende informatie in de fabula aan om de strip beter weer te kunnen geven (eis 2b). Een voorbeeld hiervan is dat het programma zelf besluit dat twee karakters naast elkaar moeten staan als ze met elkaar vechten. Er zijn nog wel een aantal andere punten waar het programma dit soort missende details nog niet zelf goed bedenkt. Een aantal van deze punten worden genoemd in de lijst met verder werk (paragraaf 8.2.2). Net als in Annie the Animator kan ons systeem omgaan met meerdere verhaallijnen (eis 2e), maar het kan voorkomen dat er veel heen en weer geschakeld wordt tussen de lijnen.

Het systeem biedt de mogelijkheid om verschillende presenters te gebruiken (eis 3a en 3b). Momenteel bestaat er maar één presenter, maar er zou dus een andere geprogrammeerd en aangekoppeld kunnen worden, bijvoorbeeld om een tekstuele representatie te maken in plaats van een grafische strip.

Secundaire functionaliteit

De secundaire functionele eisen zijn te vinden in paragraaf 3.1.2. Het toevoegen van achtergronden aan kaders is geïmplementeerd (eis 1). Extra kaders toevoegen (eis 2) is echter nog niet geïmplementeerd. Onderdelen in de cartoon boom die geen visuele representatie hebben, worden weergegeven als een rechthoek met daarin de naam van het onderdeel (eis 3).

Het is ook mogelijk om visuele representaties toe te voegen en aan te passen. Hiervoor moet wel een bepaalde structuur worden aangehouden. In de ontwikkelaarshandleiding wordt beschreven hoe dit gedaan kan worden.

Optionele functionaliteit

De optionele functionele eisen zijn te vinden in paragraaf 3.1.3. De enige eis die hieronder valt is het weergeven van de strip als een daadwerkelijk stripboek. De basis hiervoor is gelegd, maar de kaders worden momenteel nog niet in grootte aangepast aan de hand van hun belangrijkheid.

7.2.2 Non functionele eisen

Onder de non functionele eisen vallen eisen aan het platform, de software en de kwaliteit. Het platform staat los van het programma, dus wordt hier niet verder besproken. De software is inderdaad geschreven in Java en uitgebreid gedocumenteerd.

De kwaliteitseisen zijn het meest interessante onderdeel van de non functionele eisen om het programma mee te vergelijken. Het programma heeft tijdens het draaien geen gebruikersinvoer nodig, dus eis 2 is achteraf gezien niet van toepassing. Als er verkeerde invoer wordt gegeven, geeft het programma daar foutmeldingen over en sluit zichzelf af als dat nodig is (eis 3a).

Het programma is erg flexibel opgezet, zodat het inderdaad mogelijk is de presenter los van de director te vervangen (eis 4a). Alle informatie over de ontologie bevindt zich in het bestand met instellingen, zodat alleen instellingen veranderd hoeven te worden als de ontologie zou veranderen (eis 4b). Alle statusinformatie die het programma geeft is in het Engels geschreven (eis 4c). Het is mogelijk hier een andere taal voor te nemen, maar daarvoor moeten dan wel redelijk wat zinnen in allerlei klassen veranderd worden.

Eis 5a is voldaan doordat de presenter ook los gebruikt kan worden om strips te renderen. Hiervoor is het wel nodig dat er een nieuwe CartoonLoader geschreven wordt die een cartoon boom oplevert.

Het systeem is gemakkelijk te gebruiken (eis 6). In principe werkt het geheel voor het meegegeven voorbeeldverhaal meteen, zonder dat er nog iets ingevoerd moet worden. Als een gebruiker veel instellingen wil wijzigen, wordt het gebruik wel iets ingewikkelder. In de gebruikers- en ontwikkelaarshandleiding wordt beschreven waar welke instellingen voor gebruikt kunnen worden.

Hoofdstuk 8

Aanbevelingen en Verder Werk

In dit hoofdstuk worden een aantal aanbevelingen betreffende het Virtual Storyteller project gedaan en wordt er aangegeven in welke richting verder werk volgens ons het best kan gaan.

8.1 Aanbevelingen

Omdat we nog niet veel met het Virtual Storyteller project hebben gedaan, hebben we een andere kijk op het totale project. Dit geeft ons de mogelijkheid om als 'buitenstaanders' observaties te doen die lastiger zijn van binnenuit het project.

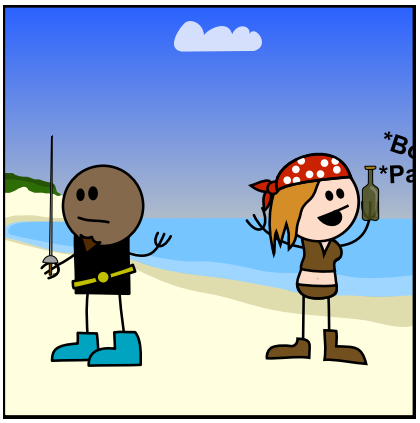
Het probleem is dat het momenteel niet mogelijk is om de wereldstaat op een bepaald tijdstip in het verhaal op te vragen. Dit probleem kan opgelost worden door één van de volgende drie aanbevelingen te implementeren:

1. Een standaardinterface voor het verkrijgen van de wereldstaat,
2. De gevolgen van acties in de fabula opnemen of
3. De gevolgen van acties extern definiëren.

Standaardinterface voor wereldstaat Het zou voor alle projecten die verder werken op de uitvoer van de Virtual Storyteller handig zijn om een standaardinterface te hebben die gebruikt kan worden om de staat van de wereld op een bepaald tijdstip op te vragen. Het idee is dat er een gedefinieerde interface is waarmee de wereldstaat op een bepaald tijdstip in het verhaal op te vragen is, bijvoorbeeld als een Jena graaf. Deze standaardinterface heeft nog steeds een tussenformaat nodig.

De gevolgen van acties in de fabula opnemen Als alle gevolgen van acties in de fabula aanwezig zijn, dan kan de wereldstaat op een bepaald tijdstip in het verhaal worden bepaald, door deze veranderingen toe te passen op de wereldstaat.

De gevolgen van acties extern definiëren De gevolgen van acties zijn in de Virtual Storyteller strikt gedefinieerd. Hier hebben andere applicaties echter geen toegang tot. Een mogelijkheid is om deze gevolgen op formele wijze vast te leggen op een zodanige manier dat deze makkelijk in te lezen is voor andere applicaties. Op deze manier kan ook de wereldstaat op een bepaald tijdstip in het verhaal worden bepaald.



Figuur 8.1: Voorbeeld van twee piraten die elkaar aanvallen

8.2 Verder Werk

Onze aanbevelingen voor de verdere ontwikkeling van Carrie the Cartoonist worden in deze paragraaf beschreven.

8.2.1 SpaceTimeAnalyser

Als dit project als basis gebruikt wordt voor verder uitbreidingen, dan is het van belang dat de SpaceTimeAnalyser uit de director compleet vervangen wordt door een component dat de staat van de wereld op een betere manier bepaalt. Zoals het er nu voor staat, kan de BaseSpaceTimeAnalyser geen veranderende staat verwerken zoals beschreven in paragraaf 5.2.4.

Hoewel de gevolgen van acties niet aanwezig in onze voorbeeldverhalen, schijnen ze wel als percepties te worden vastgelegd in de uitvoer van de Virtual Storyteller. In dit geval is er dus geen objectieve werkelijkheid, maar wordt de werkelijkheid vastgelegd door wat de karakters waarnemen. Dit is dus een voor ons minder geschikte implementatie van optie 2 zoals beschreven in de vorige paragraaf.

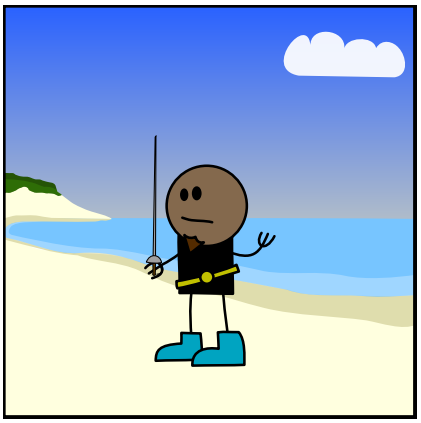
Ons voorstel is om de SpaceTimeAnalyser te vervangen door een variant die wel de wereldstaat aanpast als er een actie wordt gedaan. Dit dient op een zodanige manier te worden gedaan dat de acties niet opnieuw geïmplementeerd hoeven te worden. Door de percepties op een correcte wijze te gebruiken zou dit mogelijk moeten zijn.

8.2.2 Visualisatie

Hierna is het van belang de expressiekracht van de visualisatie te verbeteren. Het belangrijkste drietal punten is:

- Kijkrichting karakters,
- Acties en
- Frame lay-out.

Figuur 8.2: Voorbeeld van drinkende piraat



Figuur 8.3: Voorbeeld van piraat die het kader afloopt

Kijkrichting karakters Een grote stap in de goede richting is het bijhouden welke kant een entiteit op kijkt. Dit moet zowel in het kader lay-out algoritme als bij elke resource worden bijgehouden. Als dit is geïmplementeerd dan is er veel duidelijker te zien welke entiteit met welke andere bezig is, wat in afbeelding 8.1 slecht te zien is, en of iemand het kader op of af komt.

Acties De actievisualisaties hebben momenteel geen mogelijkheid om de andere items te beïnvloeden. Een voorbeeld hiervan is het leegdrinken van een fles. Zoals in afbeelding 8.2 te zien is, blijft de fles tijdens het drinken rechtop. Het zou duidelijker zijn als de fles kantelt.

Als de actievisualizers toegang zouden hebben tot het rendering subsysteem kunnen ze een betere weergave maken van een actie die wordt uitgevoerd.

Frame lay-out Het lay-out algoritme dat wordt gebruikt om de verschillende entiteiten op een frame te zetten is erg beperkt. Grote verbetering kan bereikt worden door entiteiten op verschillende dieptes te plaatsen. Door een maximaal aantal entiteiten toe te staan op de voorgrond en de rest van de entiteiten op het kader op de achtergrond te plaatsen, is er ruimte voor meer entiteiten en kunnen de entiteiten die in dit kader actief zijn op de voorgrond geplaatst worden.

Een tweede verbetering is de lay-out gebruik laten maken van een minder beperkt algoritme om te bepalen hoe de karakters naast elkaar staan. Nu worden de karakters gelijkmatig over het kader verdeeld. Afbeelding 8.3, waarin een piraat van het kader afloopt, is hier een goed voorbeeld van. De piraat staat aan de rand van alle entiteiten, maar omdat er slechts één entiteit is, wordt de piraat in het midden van het kader geplaatst. Hierdoor wordt niet duidelijk dat de piraat wegloopt van de locatie in het kader.

Hoofdstuk 9

Nawoord

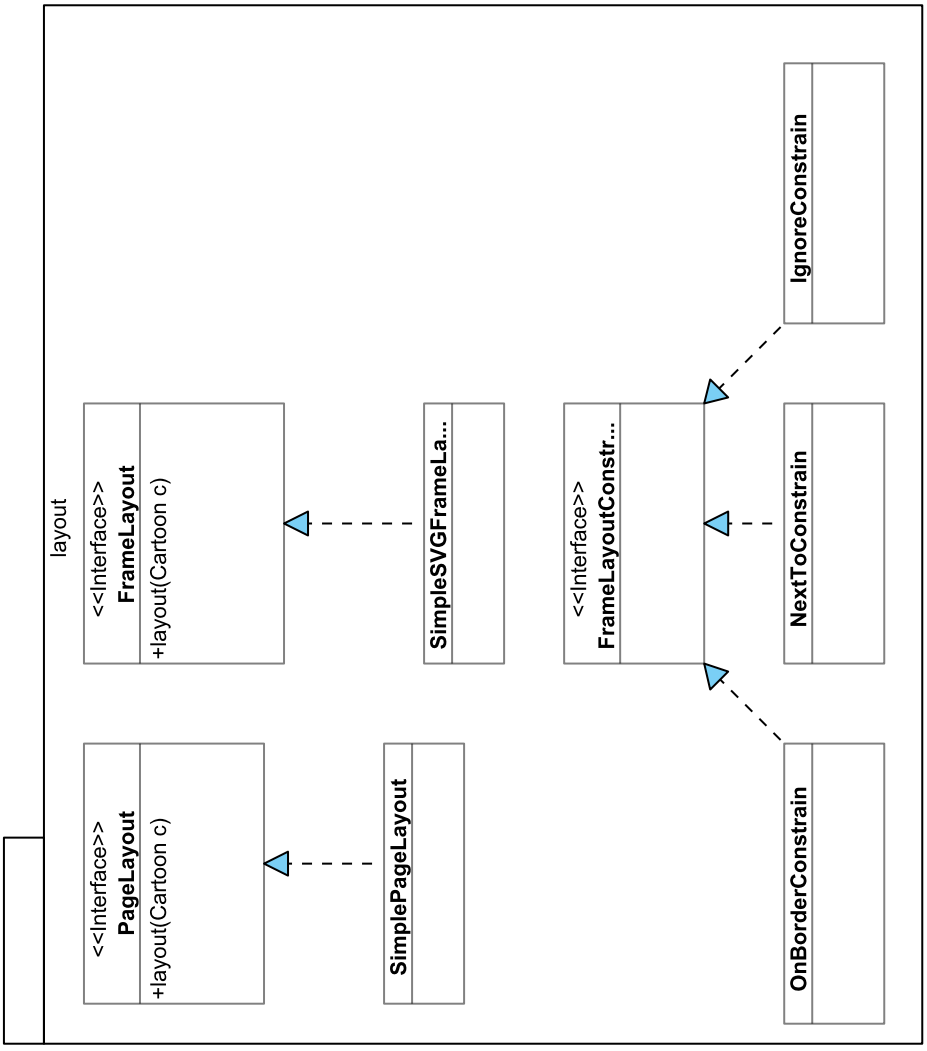
Hoewel er nog veel gedaan kan worden aan Carrie the Cartoonist, zijn we erg tevreden met wat we bereikt hebben tijdens dit project. Er kunnen voor een beperkt verhaaldomein al behoorlijk goede stripverhalen worden gegenereerd. Het programma is zo geschreven dat vrijwel alles instelbaar is, wat veel flexibiliteit geeft. Verschillende onderdelen zijn los herbruikbaar. Dit soort dingen geven het resultaat nog net wat extra meerwaarde.

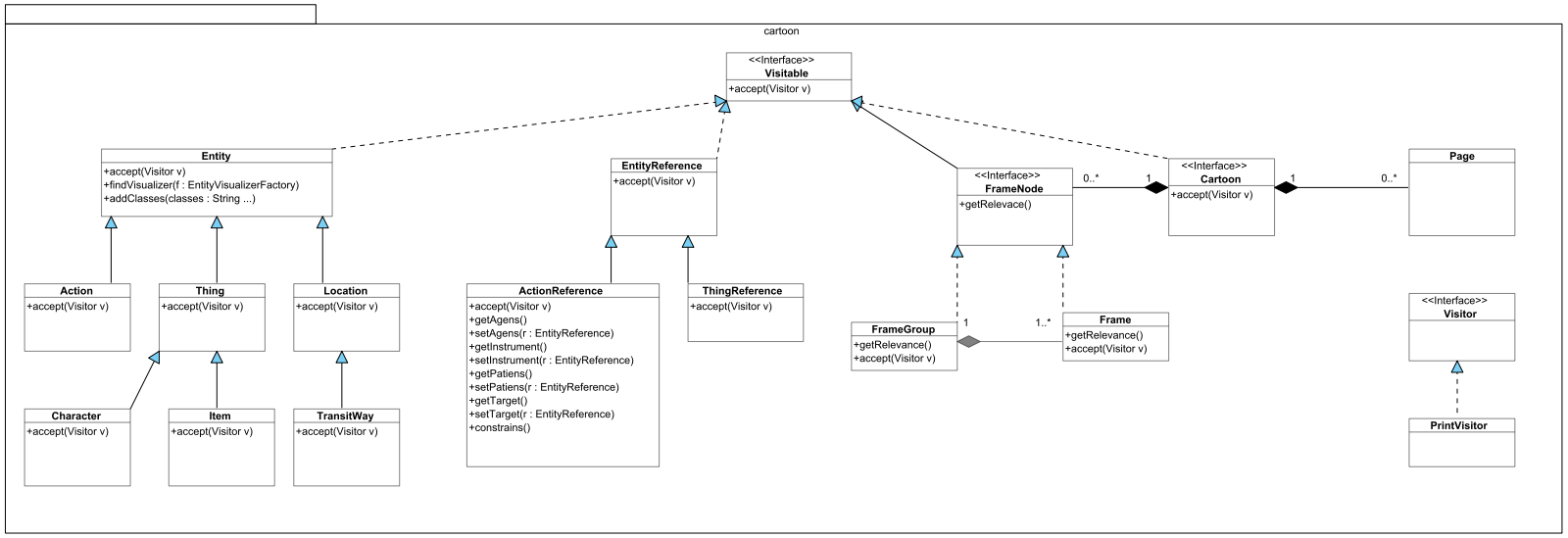
Een belangrijk leerpunt in dit project was, dat het toch veel moeilijker is om verder te bouwen op andermans code dan we van tevoren gedacht hadden. Als we dit van te voren hadden voorzien, zouden we liever alles zelf hebben geschreven in plaats van verder te bouwen op een bestaand systeem.

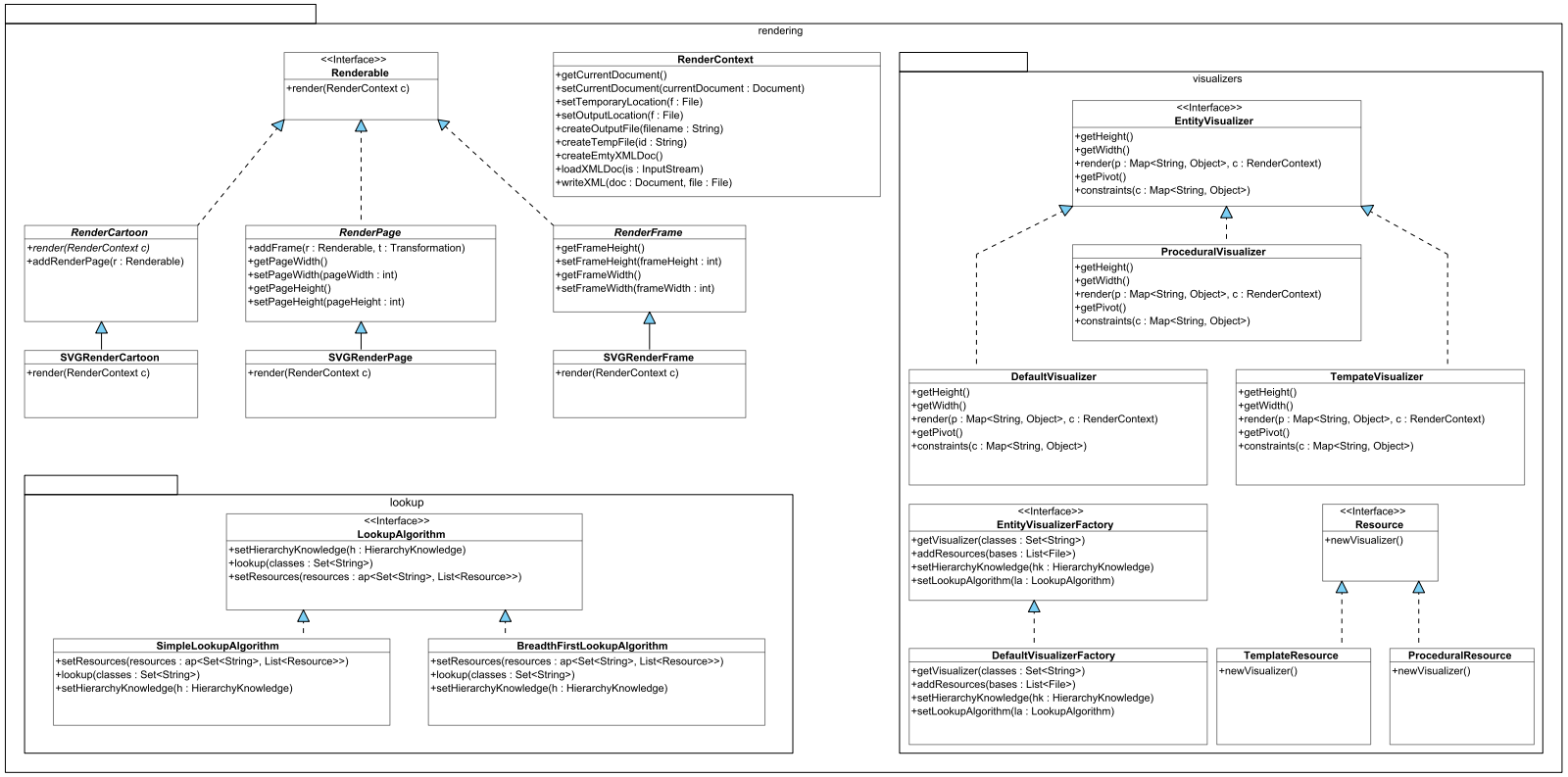
Het werken in een team ging in het algemeen goed. We hebben in het begin één keer onze planning aan moeten passen, omdat het begrijpen van de bestaande code meer tijd kostte dan verwacht. Daarna is alles netjes volgens planning gelopen. Het was erg prettig om aan het einde van het project nog de ruimte te hebben om nog wat laatste puntjes op de i te zetten, in plaats van de laatste nacht voor de deadline nog bezig te zijn met verslag schrijven.

Bijlage A

Klassen diagram




cartoon
<<Interface>>
Visitable
+accept(Visitor v)
Entity
+accept(Visitor v)
+findVisualizer(f : EntityVisualizerFactory)
+addClasses(classes : String ...)
EntityReference
+accept(Visitor v)
<<Interface>>
FrameNode
+getRelevace()
0..*
1
<<Interface>>
Cartoon
+accept(Visitor v)
1
0..*
Page
Action
+accept(Visitor v)
Thing
+accept(Visitor v)
Location
+accept(Visitor v)
ActionReference
+accept(Visitor v)
+getAgens()
+setAgens(r : EntityReference)
+getInstrument()
+setInstrument(r : EntityReference)
+getPatiens()
+setPatiens(r : EntityReference)
+getTarget()
+setTarget(r : EntityReference)
+constrains()
ThingReference
+accept(Visitor v)
FrameGroup
+getRelevance()
+accept(Visitor v)
1
1..*
Frame
+getRelevance()
+accept(Visitor v)
<<Interface>>
Visitor
Character
+accept(Visitor v)
Item
+accept(Visitor v)
TransitWay
+accept(Visitor v)
PrintVisitor


rendering
<<Interface>>
Renderable
+render(RenderContext c)
RenderContext
+getCurrentDocument()
+setCurrentDocument(currentDocument : Document)
+setTemporaryLocation(f : File)
+setOutputLocation(f : File)
+createOutputFile(filename : String)
+createTempFile(id : String)
+createEmtyXMLDoc()
+loadXMLDoc(is : InputStream)
+writeXML(doc : Document, file : File)
RenderCartoon
+render(RenderContext c)
+addRenderPage(r : Renderable)
RenderPage
+addFrame(r : Renderable, t : Transformation)
+getPageWidth()
+setPageWidth(pageWidth : int)
+getPageHeight()
+setPageHeight(pageHeight : int)
RenderFrame
+getFrameHeight()
+setFrameHeight(frameHeight : int)
+getFrameWidth()
+setFrameWidth(frameWidth : int)
SVGRenderCartoon
+render(RenderContext c)
SVGRenderPage
+render(RenderContext c)
SVGRenderFrame
+render(RenderContext c)
lookup
<<Interface>>
LookupAlgorithm
+setHierarchyKnowledge(h : HierarchyKnowledge)
+lookup(classes : Set<String>)
+setResources(resources : ap<Set<String>, List<Resource>>)
SimpleLookupAlgorithm
+setResources(resources : ap<Set<String>, List<Resource>>)
+lookup(classes : Set<String>)
+setHierarchyKnowledge(h : HierarchyKnowledge)
BreadthFirstLookupAlgorithm
+setResources(resources : ap<Set<String>, List<Resource>>)
+lookup(classes : Set<String>)
+setHierarchyKnowledge(h : HierarchyKnowledge)
visualizers
<<Interface>>
EntityVisualizer
+getHeight()
+getWidth()
+render(p : Map<String, Object>, c : RenderContext)
+getPivot()
+constraints(c : Map<String, Object>)
ProceduralVisualizer
+getHeight()
+getWidth()
+render(p : Map<String, Object>, c : RenderContext)
+getPivot()
+constraints(c : Map<String, Object>)
DefaultVisualizer
+getHeight()
+getWidth()
+render(p : Map<String, Object>, c : RenderContext)
+getPivot()
+constraints(c : Map<String, Object>)
TempateVisualizer
+getHeight()
+getWidth()
+render(p : Map<String, Object>, c : RenderContext)
+getPivot()
+constraints(c : Map<String, Object>)
<<Interface>>
EntityVisualizerFactory
+getVisualizer(classes : Set<String>)
+addResources(bases : List<File>)
+setHierarchyKnowledge(hk : HierarchyKnowledge)
+setLookupAlgorithm(la : LookupAlgorithm)
<<Interface>>
Resource
+newVisualizer()
DefaultVisualizerFactory
+getVisualizer(classes : Set<String>)
+addResources(bases : List<File>)
+setHierarchyKnowledge(hk : HierarchyKnowledge)
+setLookupAlgorithm(la : LookupAlgorithm)
TemplateResource
+newVisualizer()
ProceduralResource
+newVisualizer()

Bijlage B

Voorbeeldfabula

```
@prefix rdf:         <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:       <http://www.w3.org/2000/01/rdf-schema#> .

@prefix swc:         <http://www.owl-ontologies.com/StoryWorldCore.owl#> .
@prefix fk:           <http://www.owl-ontologies.com/FabulaKnowledge.owl#> .

@prefix ps:           <http://www.owl-ontologies.com/StoryWorldSettings/Pirates#> .
@prefix cs:           <http://www.owl-ontologies.com/StoryWorldSettings/CarrieStories#> .
@prefix :              <http://tempest.student.utwente.nl/~carrie/fabulae/pirates-1#> .

:main {
#setup (locaties)
        # De caribische zee
        :sea1 a ps:Sea ;
                swc:hasRegion :archipelago1 .

        # De 'Black stone' archipelago
        :archipelago1 a swc:GeographicArea ;
                swc:hasRegion :island1 ;
                rdfs:label "Black stone archipelago" .

        # Blood cliff isle
        :island1 a ps:Island ;
                rdfs:label "Blood Isle" .
        :island1_inland a ps:Inland ; swc:partOfGeographicArea :island1 .
        :island1_beach1 a ps:Beach ; swc:partOfGeographicArea :island1 .
        :island1_beach2 a ps:Beach ; swc:partOfGeographicArea :island1 .

        #verbind   inland en beach1
        :beach_path_a a swc:GroundWay ;
                swc:fromGeographicArea :island1_beach1 ;
                swc:toGeographicArea :island1_inland .
        :beach_path_b a swc:GroundWay ;
                swc:fromGeographicArea :island1_inland ;
                swc:toGeographicArea :island1_beach1 .

        #verbind beach1 en beach2
        :beach_connector_a a swc:GroundWay ;
                swc:fromGeographicArea :island1_beach1 ;
                swc:toGeographicArea :island1_beach2 .
        :beach_connector_b a swc:GroundWay ;
                swc:fromGeographicArea :island1_beach2 ;
                swc:toGeographicArea :island1_beach1 .

        #verbind sea1 met beach2
        :beach_connector_c a swc:WaterWay ;
                swc:fromGeographicArea :sea1 ;
                swc:toGeographicArea :island1_beach2 .
        :beach_connector_d a swc:WaterWay ;
                swc:fromGeographicArea :island1_beach2 ;
```

```
                swc:toGeographicArea :sea1 .

#setup (cast en items)
        #Sloep
        :sloop a ps:Boat ;
                swc:locatedAt :island1_beach2 ;
                rdfs:label "Een sloep" .
        #Rum
        :rum a ps:Rum ;
                rdfs:label "Rum iemand?" ;
                swc:containedBy :bottle_o_rum .  # contains
        :bottle_o_rum a ps:Bottle ;
                rdfs:label "Een fles." ;
                swc:contains :rum .

        #Rapier
        :rapier1 a ps:Rapier ;
                rdfs:label "Een soort zwaard!" .

        #Cast
        :pirate1 a cs:FemalePirate ;
                swc:holds :bottle_o_rum ; # holds
                swc:locatedAt :island1_inland .

        :pirate2 a cs:MalePirate ;
                swc:holds :rapier1 ; # holds
                swc:locatedAt :island1_beach2 .

# Officieel moet overal een fk:character bij, maar volgens de spacetime analyzer mag een character
# op een groundway zijn.  Hierdoor zal de locatie van de scene worden ingesteld op de groundway.
# De spacetime analyzer zal eerst herschreven moeten worden voor dat de fk:character relaties
# toegevoegd kunnen worden.

# Eigenlijk moet er een datatype worden toegevoegd aan de time literal.  Dit zorgt echter voor
# fouten in de Named Graphs 4 Jena package.
# ("1000"^^<http://www.w3.org/2001/XMLSchema#int> i.p.v.  "1000")

#testscene
        # piraat1 loopt van inland naar beach1
        :action10 a fk:Walk ;
                #fk:character :pirate1 ;
                fk:agens :pirate1 ;
                fk:instrument :beach_path_b ;
                fk:target :island1_beach1 ;
                fk:time   "1000" .

        # piraat2 loopt van beach2 naar beach1
        :action20 a fk:Walk ;
                #fk:character :pirate2 ;
                fk:agens :pirate2 ;
                fk:instrument :beach_connector_b ;
                fk:target :island1_beach1 ;
                fk:time   "2000" .

        # Piraat2 valt Piraat1 aan
        :action30 a fk:Strike ;
                #fk:character :pirate2 ;
                fk:agens :pirate2 ;
                fk:instrument :rapier1 ;
                fk:patiens :pirate1 ;
                fk:time   "3000" .

        #piraat1 drinkt de rum
        :action31 a fk:Drink ;
                #fk:character :pirate1 ;
                fk:agens :pirate1 ;
                fk:patiens :bottle_o_rum ;
                fk:target :rum ;
                fk:time "3001" .

        # Piraat1 valt piraat2 aan
```

```
        :action32 a fk:Strike ;
                #fk:character :pirate1 ;
                fk:agens :pirate1 ;
                fk:instrument :bottle_o_rum ;
                fk:patiens :pirate2 ;
                fk:time   "3002" .

        # Piraat 1 vindt het leuk om Piraat 2 aan te vallen
        :emotion1 a fk:Joy;
                fk:character :pirate1 ;
                fk:time "3002" .

        #piraat 1 vindt het niet leuk om geslagen te worden
        #:emotion2 a fk:Sadness;
        #      fk:character :pirate1 ;
        #      fk:time "3000" .

}
```

Bijlage C

Cartoonscript voorbeeld

C.1 Inleiding

Het was in eerste instantie de bedoeling om een zeer strikte scheiding tussen de Director en de Presenter zijde te realiseren. Hiervoor is het cartoonscript ontworpen. De Director creëerd het cartoonscript en de Presenter leest het cartoonscript in. Hierdoor kan men gemakkelijk één van de twee delen herschrijven zonder dat dit gevolgen heeft voor het andere deel. Tijdens de ontwikkeling hebben wij echter besloten om de XML serialisatie en deserialisatie achterwege te laten omdat wij prioriteit hebben gegeven aan het implementeren van meer mogelijkheden in de Presenter.

C.2 Voorbeeld

```
<?xml  version = "1.0"  encoding = "utf-8" ?>
<!  DOCTYPE  cartoonscript  [
    <!  ENTITY  swc  "http://www.owl-ontologies.com/StoryWorldCore.owl#" >
   <!  ENTITY  fk  "http://www.owl-ontologies.com/FabulaKnowledge.owl#" >
]>
<cartoonscript  xmlns = "cartoonist:cartoonscript"  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns"
>
        <declarations>
                <character  id = "grandma" >
                        <rdf:type  rdf:resource = "&fk;Grandma"  />
                </character>
                <character  id = "red" >
                        <rdf:type  rdf:resource = "&swc;Human"  />
                </character>
                <character  id = "wolf" >
                        <rdf:type  rdf:resource = "&swc;Animal"  />
                </character>

                <item  id = "sword" >
                        <rdf:type  rdf:resource = "&swc;Device"  />
                </item>
                <item  id = "basket" >
                        <rdf:type  rdf:resource = "&swc;Container"  />
                </item>
                <item  id = "apple" >
                        <rdf:type  rdf:resource = "&swc;FruitOrVegetable"  />
                </item>

                <location  id = "hut" >
                        <rdf:type  rdf:resource = "&swc;GeographicArea"  />
                </location>
                <location  id = "forrest" >
                        <rdf:type  rdf:resource = "&swc;GeographicArea"  />
                </location>

                <transitway  a = "forrest"  b = "hut"  id = "path" >
                        <rdf:type  rdf:resource = "&swc;GroundWay"  />
                </transitway>
        </declarations>

        <frames>
                <frame  relevance = "1.0"  location = "forrest" >
                        <character  ref = "red" >
                                <mood  rdf:resource = "&fk;Joy" />
                                <carries>
```

```
                                <item  ref = "basket" >
                                        <contains>
                                                <item  ref = "apple"  />
                                        </contains>
                                </item>
                                <item  ref = "sword"  />
                        </carries>
                </character>
        </frame>

        <frame  relevance = "0.87"  location = "hut" >
                <character  ref = "grandma"  />
        </frame>

        <framegroup>
                <frame  relevance = "1.0"  location = "forrest" >
                        <character  ref = "red" >
                                <carries>
                                        <item  ref = "basket" >
                                                <contains>
                                                        <item  ref = "apple"  />
                                                </contains>
                                        </item>
                                </carries>
                                <holds>
                                        <item  ref = "sword"  />
                                </holds>
                        </character>
                        <action>
                                <rdf:type  rdf:resource = "&fk;Take"  />
                                <agens  ref = "red"  />
                                <patiens  ref = "sword"  />
                        </action>

                        <character  ref = "wolf"  />
                        <action>
                                <rdf:type  rdf:resource = "&fk;Walk"  />

                                <agens  ref = "wolf"  />
                                <instrument  ref = "path"  />
                                <patiens  ref = "forrest"  />
                        </action>
                </frame>

                <frame  relevance = "1.0"  location = "forrest" >
                        <character  ref = "red" >
                                <carries>
                                        <item  ref = "basket" >
                                                <contains>
                                                        <item  ref = "apple"  />
                                                </contains>
                                        </item>
                                </carries>
                                <holds>
                                        <item  ref = "sword"  />
                                </holds>
                        </character>

                        <character  ref = "wolf" >
                                <mood  rdf:resource = "&fk;Sadness" />
                        </character>

                        <action>
                                <rdf:type  rdf:resource = "&fk;Strike"  />
                                <agens  ref = "red" />
                                <patiens  ref = "wolf" />
                                <instrument  ref = "sword" />
                        </action>
                </frame>
        </framegroup>

    </frames>
</cartoonscript>
```

Verklarende Woordenlijst

Cartoonscript Het tussenformaat tussen de director en presenter. Dit formaat wordt intern gebruikt, 3

CSS Cascading Style Sheets is een techniek voor de stijl (vormgeving) van webpagina's, 12

Director Het deel van het systeem dat een plot vertaalt naar cartoonscript, 3

Fabula hetgeen in de verhaalwereld bestaat en gebeurt en waarom, 4

OWL Een W3C-standaard voor het definiëren van Webontologieën, 5

PDF Het Portable Document Format is een de facto standaard voor de uitwisseling van elektronische documenten en formulieren die in hun oorspronkelijke vorm gereproduceerd moeten kunnen worden, 12

Plot een relevante selectie uit de fabula die een coherente en consistente verhaallijn oplevert, 3

PNG Portable Network Graphic is een bestandsformaat voor afbeeldingen met verliesloze compressie, 12

Presenter Het deel van het systeem dat cartoonscript omzet naar een stripverhaal, 3

SPARQL Een RDF query language, 8

SVG Scalable Vector Graphics is een op XML gebaseerde internetstandaard voor statische en dynamische vectorafbeeldingen, 12

TriG TriG is een tekstformaat voor het serialiseren van Named Graphs en RDF Datasets, 5

XML eXtensible Markup Language is een standaard voor het definiëren van formele markup-talen voor de representatie van gestructureerde gegevens in de vorm van platte tekst, 12

XSL-FO XSL Formatting Objects is dat onderdeel van de XSL specificaties dat de formattering van XML documenten voor zijn rekening neemt, 12

Bibliografie

[1] Apache Formatting Objects Processor, Oct 2007.

[2] Batik SVG Toolkit, Nov 2007.

[3] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, Jan 2008.

[4] Extensible Markup Language (XML) 1.0 (Fourth Edition), Jan 2008.

[5] Extensible Stylesheet Language (XSL) Version 1.1, Jan 2008.

[6] Feikje Hielkema. Performing Syntactic Aggregation using Discourse Structures. Master's thesis, May 2005.

[7] Ivo van Hurne, Andre Loker, Isaac Pouw, Edwin Vlieg, and Ronald Volgers. Annie the Animator. 2007.

[8] Adobe Systems Incorporated. *PDF Reference third edition, Adobe Portable Document Format Version 1.4*. AddisonWesley, Dec 2001.

[9] Javadoc Tool Home Page, Sept 2007.

[10] Jena Semantic Web Framework, Nov 2007.

[11] OWL Web Ontology Language Overview, Jan 2008.

[12] Portable Network Graphics (PNG) Specification (Second Edition), Jan 2008.

[13] Scalable Vector Graphics (SVG) 1.1 Specification, Jan 2008.

[14] I. Swartjes and M. Theune. A Fabula Model for Emergent Narrative. *Technologies for Interactive Digital Storytelling and Entertainment, Third International Conference, TIDSE 2006, Darmstadt, Germany*, 4326:49–60, 2006.

[15] The TriG Syntax, Jan 2008.