June 6, 2014

MASTER'S THESIS

GENERATING DUTCH FOCALIZED STORIES ABOUT INTERACTION IN A SERIOUS GAME.

Marissa Hoek

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Human Media Interaction

Exam committee:
dr. M. Theune
dr. ir. H. J. A. op den Akker
J. M. Linssen MSc

UNIVERSITY OF TWENTE.

Abstract

Currently a serious game is being developed by the Human Media Interaction group of the University of Twente, in cooperation with T-Xchange, a company that specializes in serious games. This game is based on the Virtual Storyteller, a multi-agent system that is used to simulate interaction between virtual characters, and designed to train the social skills of police officers.

To learn from this serious game, it is important for players to reflect on what has happened in the game. To support this, a report can be made be about the events of the game. In this project, I aim to automatically generate such reports in natural language. In addition, I will investigate techniques that can be used to possibly improve how much the player learns from the report.

One such technique is focalization. When training social skills, it might be very useful to get an insight in the motivations behind the actions of the other characters, as well as knowing how the other characters perceived your actions. To do this, focalization could be used. Focalization is a narratological concept, meaning looking at a story through the eyes of one of the characters. Telling the story from the perspective of one of the characters can have positive effect on how much the reader understands the character.

Another technique I investigated is flashbacks. It is possible that players will interact with the same set of characters through multiple sessions. In that case, actions from the previous sessions can influence the current game. In this case, flashbacks should be used to link the events of the previous sessions to the current.

An existing module for the Virtual Storyteller called the Narrator was used as the basis for this project. The Narrator was already able to tell neutral stories about the actions of non-player characters based on a kind of log of their interaction in the form of a causal network. However, the stories it was made to create are fairy tales. The goal of this project is thus to modify the existing Narrator to support focalization and flashbacks, and to update the design of the Narrator to work well with the new serious game.

Contents

Chapter 1

Introduction

In this report I describe my research into creating a system that can automatically generate reports in natural language on the interaction between players and virtual characters in a serious game. This serious game is currently being developed by research at the HMI group at the University of Twente, together with T-Xchange, a company that specializes in serious games. The game is part of a project called AGENT, and will be designed to train the social skills of police officers (Linssen and de Groot, 2013). The first scenario developed for this project is called “Loitering Juveniles”, and is about a police officer dealing with a group of young people causing trouble on the street.

To help players reflect on this game, we think it will be useful if they can read a report afterwards to help them reflect on their own actions, as well as understand the actions of the characters. That is the goal of this project. Besides just generating stories, I investigate techniques that could improve how much the reader learns from the report, by helping them understand the motivations behind the actions of the virtual characters.

Virtual Storyteller The new serious game is based on the Virtual Storyteller, a multi-agent system that is used to generate stories using an ‘emergent narrative’ approach (Swartjes and Theune, 2008). Emergent narrative means in this case that the system simulates the behavior of characters as agents, and that the story *emerges* from the interaction between these agents.

The Virtual Storyteller is a multi-agent system that is used to simulate interaction between characters, who are played by agents. The Virtual Storyteller has several types of agents. The first type is the Plot Agent, which leads the simulation and tries to steer the story into an interesting direction. The Plot Agent uses a World Agent to keep track of the state of the story world. The characters are ‘played’ by Character Agents. The job of these agents is to react to situations in the story world as if they are the character they play (Swartjes, 2010, Chapter 6). A Character Agent can also be controlled by a human.

The Virtual Storyteller uses turn-based simulation. Each round the Character Agents are asked in turn to select an action. The system then calculates the effects of the selected action on the story world, and asks the next Character Agent to select an action.

To make the story more interesting, the Virtual Storyteller uses improvisation techniques. Two of these techniques are out-of-character communication and late commitment. Out-of-character communication refers to a technique where characters can communicate ‘behind the scenes’. The aim is then not to achieve the character’s goal, but to make the story more interesting (Swartjes, 2010, Chapter 6). Late commitment refers to the idea of filling in story details as late as possible, to keep all options open for the characters. For example, using late commitment it’s possible to decide at the last moment that a chest contains a bottle of rum, so that the pirate captain can take the bottle from the chest and drink it.

The old Virtual Storyteller had a module called the Narrator, which transformed the events from the Virtual Storyteller into Dutch text. The Narrator was created by Nanda Slabbers (Slabbers, 2006) based on a surface realizer by Feikje Hielkema (Hielkema, 2005), and later modified by René Zeeders (Zeeders, 2008). It makes sense to use (part of) this module for creating reports for AGENT. However the old Virtual Storyteller and Narrator were made for a fairytale domain, so to create good reports the Narrator first has to be adapted to the domain of the serious game.

Apart from adapting the Narrator to this new domain, I investigate techniques that could improve how much the trainee learns from the report.

Focalization Core et al. (2006) describe the idea of explainable Artificial Intelligence, which is all about building a system that explains the causes behind what happens in a simulation. This is done in the context of a military simulation used for training soldiers. In this domain, it’s important that soldiers reflect on their actions after training. This is called an after-action review (AAR). In traditional live exercises, which happen in the field with actual equipment, an important part of this is the perspective of the instructors that play the opposing forces. According to the US field manual (Army, 1990), the

feedback from the opposing forces can provide a really valuable insight in their plans, which helps from a training perspective.

In a computer simulation however, the opposing forces are played by an artificial intelligence, as opposed to human instructors. It can therefore be very valuable to have the AI explain its actions, and the causal relationship between events in a simulation.

The AGENT domain, in which police officers partake in a simulated interaction with loitering youth, has a lot in common with a military simulation. In this case, the 'opposing forces' are the loitering adolescents. Instead of defeating them in combat, the police officer must use their social skills to persuade the loitering youths for example go to another place where they are not bothering passersby. I expect that in this case, information about the causes of the actions of the youths from their perspective can provide valuable insight and help with training.

To provide the trainee with insight on the actions of the characters, I will use a narrative technique called focalization. This technique refers to telling the story from the viewpoint of a certain character.

Flashbacks Another technique I investigate is flashbacks to previous interactions. The idea is that a trainee can interact with the characters in multiple sessions. Sometimes, the reason for an A.I. character to behave a certain way can be found in the events of a previous session. In this case, the characters have a 'back story' that influences how they currently react. A typical way of linking events in the back story to the current story is through flashbacks. A flashback is a narrative concept in which the story temporarily tells events from a previous time, instead of using chronological order.

To implement flashbacks in the new Narrator I use a technique which links the emotions of characters in the current story to the causes of these emotions in the back story.

Thesis overview

In this report I present my research into creating a natural language generation (NLG) system that reports on the events of a serious game for police training. To do this I modified an existing program called the Narrator and ported it to the new domain from the serious game. To try and improve how much the trainee learns from the generated text, I have added focalization and flashbacks.

I start by looking at related work, as presented in Chapter 2. I look at similar projects and review the basic literature on the architecture of NLG systems. In Chapter 3 I analyze the existing system. I look at the existing data formats and the structure of the Narrator, and look at improvements that can be made.

In Chapters 4 and 5 I present how I made focalization and flashbacks respectively. In both chapters, I start by introducing the concept. Then I look at related work and present my methods. In Chapter 6 I describe in detail the new data formats that are used in the new system and in Chapter 7 I describe in detail how the system works.

In Chapter 8 I describe how I've evaluated the new system. I will also present and discuss my results. Finally, in Chapter 9, I conclude my findings and present recommendations for future work.

Chapter 2

Related work

This section describes literature that is related to the Narrator. It is split in two sections. In the first I survey literature that describes the different basic architectures of systems that generate natural language. In the second section I describe systems that are similar to the Narrator or the Virtual Storyteller, i.e. NLG systems that can write stories, or A.I. systems that create the events of such a story.

2.1 Architecture

This section describes the basic literature regarding the architecture of NLG systems.

Reiter and Dale

The book by Reiter and Dale (2000) is seen as the standard work about creating natural language generation systems. It discusses the specific tasks a NLG system should perform, and proposes a three-stage architecture for such a system. The focus of the book is on making simple and practical systems that can be built with relative ease.

Reiter and Dale's architecture is based on a pipelined structure with three parts: A high level *Text Planner*, a *Sentence Planner* and a *Linguistic Realizer*. The text planner performs content determination, which is selecting the information that should be present in the final text. Often the data that should be converted to text has a lot of excess information, in this step the NLG system selects what should be communicated. The text planner also performs discourse planning. In this step the system decides how the data should be structured in the text. The output is a text plan, which contains information about the organization and order of the data.

This text plan is then used as input for a sentence planner. This part has three subtasks: sentence aggregation, lexicalization and referring expressions generation. During the sentence aggregation phase the system decides how to combine pieces of information into a sentence. The output is a new text plan, made of sentences which contain one or more pieces of information. The planner also decides what 'syntactic mechanism', for example conjunction, ellipsis, relative clause, is used to combine the pieces of information.

The next step is lexicalization. During this phase the NLG system decides what specific words are going to be used for each entity. It chooses for example which verbs are used, and how relations are realized. During this step the NLG system also has to create referring expressions, which are the descriptions for entities. This is harder than it sounds at first, because we have to make sure that the reader understands which entity is referred to, while making the description fluent, and not too complicated and hard too read at the same time.

The final step is linguistic realization. At this step the NLG system takes the words, verbs and descriptions made in the last step and turns them into grammatically correct text. This step inflects the verbs, chooses the right article, makes sure everything is in the right grammatical case and that the words grammatically agree with each other.

Hayes and Flower

Hayes and Flower (1986) provide a different architecture for natural language projects that is based on the way humans write. It is originally intended for teaching students how to write better, but it could be applied to computer systems as well. Hayes and Flower define three processes: planning, sentence generation and revising. In the planning part, a hierarchy of goals is created. The writer first defines the main goals of the story, and then defines subgoals. These subgoals can then also have subgoals, which creates a hierarchical structure.

Although the subprocesses are similar to the three subsystems listed by Reiter and Dale, there is a big difference between the two approaches in the sense that Hayes and Flower propose an interwoven structure. This is more similar to the way a human writer would work. Sometimes a writer will create part of a story, and then another part, doing each process twice or more often. And other times, a writer would add content in the middle of a story and then recreate the goals and plans for the entire story.

RAGS

Mellish et al. (2006) describe an abstract architecture for Natural Language Generation Systems, with the goal of enabling sharing and re-use of components of such systems. They found that even though all 19 of the systems they surveyed used Reiter and Dale (2000)'s three-stage model, there were big differences in what task was handled by which stage. The systems also lacked a clear unified definition of what the intermediate data structures look like. This makes sharing of components even more difficult.

These problems are there because Reiter and Dale's architecture is not a strict manual for building NLG systems, but rather a 'consensus architecture', a description of how many NLG systems work. To counter this, Mellish et al. (2006) tried to make a reference architecture that systems could adhere to that has more definition than Reiter and Dale's consensus architecture. This resulted in the following points:

- A high level specification of internal data types.
- A low level reference implementation of an internal data model
- An XML specification of the intermediate data structures
- A generic view of how components should interact with each other
- A few reference implementations of the architecture.

Cahill et al. (2001) introduced a more concrete architecture as an implementation of the RAGS specification. This is illustrated with a system called RICHES, an actual NLG system built according to their architecture.

2.2 Similar Projects

This section describes related systems that perform a role similar to the Narrator or the Virtual Storyteller. This means natural language generation systems that focus on creating stories in natural language, or artificial intelligence systems that focus on creating the events of a story.

Curveship

Curveship (Montfort, 2011) is a framework for generating interactive fiction. This framework is interesting because it is capable of changing the narrative style. Curveship supports focalization of different actors, and telling events out of order. Curveship has two main parts: the 'Simulator' and the 'Teller'. The Simulator is the part where the story happens: it maintains the state of the world and determines whether the player's actions succeed. It then passes on a first-order representation of the events that have happened, which the Teller then uses to narrate the story.

The Teller is based on the standard three stages structure as defined by Reiter and Dale (2000). In the high-level planner, the order of narration of actions is determined. By changing this order, it's possible to tell the story in different styles. For example, chronological order, reverse order, flashbacks and flash-forwards are supported using this method.

STORYBOOK

Callaway and Lester (2002) describe STORYBOOK, their system for automatically generating narrative prose, or more specifically fairy tales. The structure of STORYBOOK is similar to the 'standard' three-stages from Reiter and Dale (2000), but with the addition of a few new modules. These modules are the discourse history, lexical choice module, revision module and narrative formatting module. They mention that none of these modules are new ideas, but that STORYBOOK is the first complete text generator to include all of them. A diagram of the structure can be found in Figure 2.1.

The system starts with the *Narrative Planner*, which creates the story plot, including characters, based on parameters. The output of this planner is the Fabula. The Fabula is passed on to the *Narrative Organizer*, a part that is comparable to Reiter and Dale's Text Planner. The narrative organizer is responsible for segmenting the story into paragraphs, and structuring the narrative. It also includes the *Discourse History*, a module that includes an ontology which is used to mark words as definite (*the*)

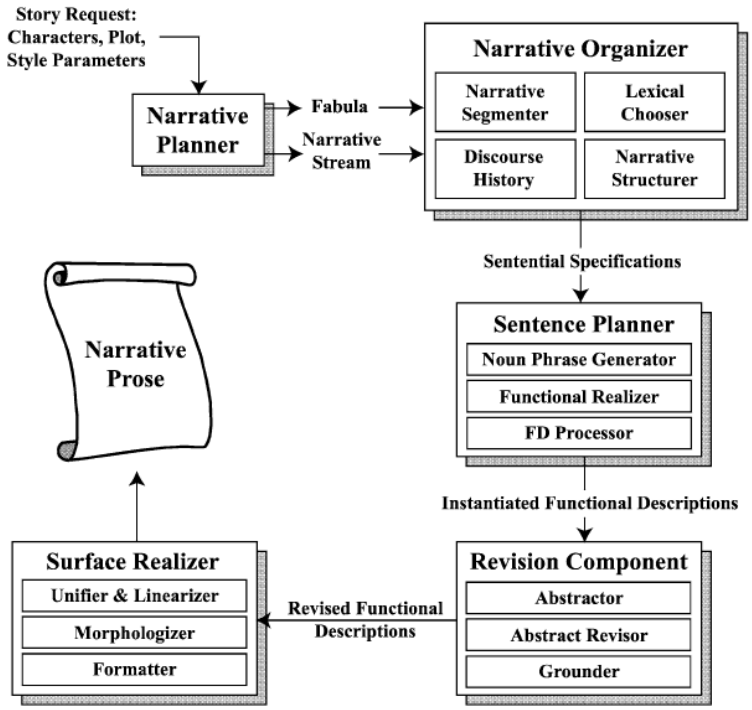

Figure 2.1: The basic structure of STORYBOOK (Callaway and Lester, 2002, Figure 7).

or indefinite (*a*), and for generating pronouns for concepts. Another module in the organizer is the lexical choice module, which checks the discourse history for repetitive use of language and changes the lexicalization to alternative words for a concept if necessary.

The second part is the *Sentence Planner*, which converts the narrative into linguistic structures known as *functional descriptions* (FDs). These FDs are a hybrid of semantic and syntactic entities and are used as input to the surface realizer. What's unique about STORYBOOK is that it includes a *Revision* module after the Sentence Planner. This revision module works by converting the sentence plan to an abstract plan containing only the essential aspects of the plan, and performing revision operators such as sentence aggregation on this abstract plan. After performing these operators, the abstract plan is then 'grounded', by adding linguistic details again so it can be used as input for the surface realizer. STORYBOOK uses the FUF surface realization system (Elhadad, 1993) as an off-the-shelf surface realizer.

Callaway and Lester (2002) have formally evaluated their system, I describe this evaluation in Section 8.1.

2.3 Discussion

The related projects that I have found both use an architecture similar to the one proposed by Reiter and Dale (2000). STORYBOOK does include a revision component as Hayes and Flower (1986) suggest, this module however performs a task that is more similar to sentence aggregation than to revising the content of part of the story. The basic structure of STORYBOOK still contains the basic three stages from Reiter and Dale. These three stages are also present in the existing version of the Narrator.

It makes sense that a lot of NLG projects use Reiter and Dale's structure, as Hayes and Flower describe a method that's more fit for a creative process where the content still has to be created. When the content is already known, it makes more sense to use the systematic pipeline approach described by Reiter and Dale. This is the case for the Narrator as well, since it does not generate the actual plot of the story: this is done by the Virtual Storyteller.

While I think standardization is a good idea, I found that RAGS is not used very often. This lack of other projects that follow the standard makes that RAGS is not very useful for the new Narrator, as it makes it not very likely that off-the-shelf modules that could possibly be included in the Narrator follow RAGS.

Chapter 3

Analysis

3.1 Introduction

In this chapter I analyze the existing system. I start by looking at the data structures used in the Narrator, as they are essential to understanding the internal workings of the system later on. I start by discussing the Fabula Model in Section 3.2, the causal network that serves as the input to the Narrator. I will describe the existing Fabula, look at related work, then look at how it can be improved. In Section 3.3, I then describe the Rhetorical Dependency Graph (RDG), the structure that is used internally as the input for the Surface Realizer, the part of the Narrator that generates the final text.

In Section 3.4, I describe the structure of the existing Narrator. Finally, in Section 3.5 I look at ways in which the Narrator can be improved. As flashbacks and focalization are discussed separately, I will look at smaller modifications.

3.2 Fabula Model

3.2.1 Current Fabula model

The Fabula model is a formal representation of the events in the story generated by the Virtual Storyteller (Swartjes and Theune, 2006). The name comes from Bal (1997)'s definition of a Fabula, which is 'a series of logically and chronically related events'. The Fabula model follows the structure of a causal network. This is based on a theory (Trabasso et al., 1984), which states that humans understand the relation between events in a story in the form of a network in which some events cause other events.

Trabasso proposed a model called the General Transition Network. This GTN consists of six different story elements (Setting, Event, Internal Response, Goal, Attempt and Outcome) which are connected by four different causal relationships: physical causality, psychological causality, motivation, and enablement.

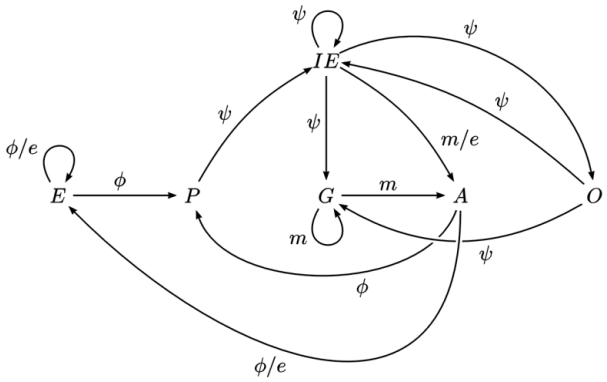

Figure 3.1: An example of a Fabula graph. The capital letters represent the different types of story events, and the arrow represent the causal relationships between these events. (Swartjes and Theune, 2006)

Although the Fabula model is based on this model, it does not follow it exactly (Swartjes, 2010, Chapter 7). One difference is that Trabasso's model is different for each character in the story, whereas the Fabula model stores one 'objective reality' of all events. Telling the story from the point of view of one character is then the responsibility of the Narrator. Another difference is that the Fabula model does not contain the Setting (e.g. characters and locations) of the story. The Fabula model also adds

Perceptions, which denote that a certain character has perceived an event. This was not necessary in the GTN as that model was different for each character. Because the Fabula model makes one global structure, it's important to store what events each character has perceived.

This leaves the Fabula with six element types, which are as follows:

Goal The drive for a character to act.

Action A world change caused by a character.

Event A world change not caused by a character.

Outcome The result of a goal. Outcomes are personal for each character, and are based on the characters beliefs instead of the actual world state.

Perception A character perceiving another story element.

Internal Element Something that happens within the character, examples are cognitions, emotions, feelings and beliefs. If a character is made angry by a certain action or event, this 'angry' emotion is an example of an Internal Element.

Apart from these six main element types, there are also four subtypes for the Goal element, which are as follows:

Attain Goal A character wants something that is not true to become true, i.e. they want something to happen. Or a character wants to possess a certain object that they do not already possess.

Avoid Goal A character wants something that is not true to *not* become true, i.e. they want to stop something from happening. Or a character does not want to possess a certain object.

Leave Goal A character wants something that is true to become not true, i.e. they want something to stop happening. Or they want to not have a certain object anymore.

Sustain Goal A character wants something that is true to stay true, i.e. they want something to keep happening. Or they want to keep a certain object.

The four causal relationships are as follows:

Physical causality An Action or Event physically causing an event. For example, in the story of Little Red Riding Hood, if the wolf is killed by the hunter's knife, the death of the wolf is physically caused by the action of the hunter cutting the wolf. If a Perception is caused by an Action or Event, this is also a physical causality.

Psychological Causality Something resulting from reasoning in the minds of characters. This can be a Perception causing an Internal Element, an Internal Element causing another Internal Element, or an Internal Element causing a Goal.

Motivation One story element motivating another. This can happen in one of three ways. A Goal motivating a subgoal, which has to be performed before the main goal can be achieved. A Goal motivating an Action, this happens when a character performs a certain action to achieve their goal. And finally an Action motivated by an Internal Element, for example a character running away because they are scared.

Enablement A certain element enables the occurrence of another element, but is not the cause of it. For example, if a pirate wants to drink rum, he can't drink until he has a rum bottle. The belief that he is holding a bottle of rum *enables* the Action of drinking rum, but it is caused by the Goal of wanting to drink rum.

The Fabula is saved as a set of RDF (Resource Description Framework) Named Graphs (Carroll et al., 2005). The set consists of a main graph which contains the Fabula elements and the causal relationships, saved as RDF triples (*subject:predicate:object*). The other graphs contain the contain of the elements, which is linked from the main graph using the predicate `fabula:hasContent`.The graphs are saved in the TriG format, which is a way of storing RDF Graphs in plain text using Turtle, a syntax made for RDF. The TriG syntax can be found on Bizer (2007)'s website.

3.2.2 Related work

MAKEBELIEVE

MAKEBELIEVE is a system by Liu and Singh (2002) which generates stories using a common sense knowledge base. The knowledge base describes how one event causes another event, for example 'drinking alcohol causes you to be drunk'. The system starts with a first line, which is supplied by the user. MAKEBELIEVE then uses the cause and effect descriptions from a knowledge base to generate a

John became very lazy at work. John lost his job. John decided to get drunk. He started to commit crimes. John went to prison. He experienced bruises. John cried. He looked at himself differently.

Figure 3.2: An example story from MAKEBELIEVE.

new event. This is done by matching the 'effect' of the last event to the 'cause' of an event described in the knowledge base. The system uses fuzzy matching, which means that the cause and effect do not have to match up exactly. This makes it more likely that there's a match, and simulates creativity.

The result of generating stories this way is a cause and effect structure that is similar to the Fabula model of the Narrator. The difference is that MAKEBELIEVE creates a one-dimensional chain of cause and effect, in which each event is the cause of only one other event that happens directly after the event that caused it. This is different from the Fabula model, which has a more complex causal network structure in which events can cause multiple other events. This simple structure is sufficient for MAKEBELIEVE because it uses a simple way of creating stories that only uses the last event as the cause of the next event.

Mimesis

Another way of generating stories is described by Young (2007). His paper describes a system called Mimesis, which can create interactive virtual worlds. They propose a bipartite model, which has a distinction between the structure of the story and the structure of the narrative discourse, that is, the actual telling of the story.

The story structures are created by something called the Decompositional Partial-Order Causal Link (DPOCL) planner (Young and Moore, 1994). This planner creates something called DPOCL plans which store the structure of the story. These plans are similar to the the Narrator's Fabulas. DPOCL plans contain a number of actions, which can be abstract, like a goal, or primitive, like a concrete action. Each action has a set of preconditions, which are the conditions that must be true in the world before the action can succeed. Each action also has a set of effects, which are the conditions of the world that are altered by successfully executing it. The DPOCL plan also contains a set of what they call *temporal constraints*, which define in what order certain steps should be executed, and causal links, which connect pairs of steps when an effect of one event matches a precondition of the other.

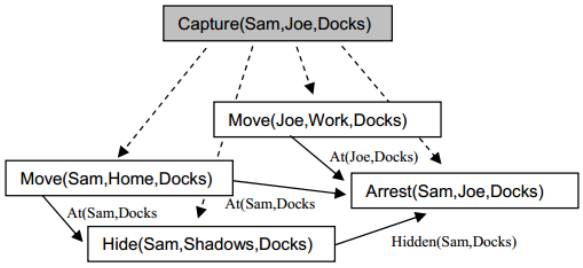

Figure 3.3: A graphical representation of a DPOCL story plan.

In figure 3.3 you can see a representation of a part of a story in DPOCL. This story is about Sam wanting to capture Joe at the docks. The grey box belongs to an 'abstract action', in this case the goal of Sam wanting to capture Joe. To attain this goal, Joe and Sam move to the docks. Sam then hides in the shadows. The preconditions for the action Arrest are now filled, and Sam can arrest Joe.

The structure of the DPOCL plans was evaluated in a research by Christian and Young (2004), which compared it to the way humans understand a story.

3.2.3 Proposed modifications

In the previous section I discussed several alternatives to the Fabula model. MAKEBELIEVE (Liu and Singh, 2002) uses a knowledge base to create a single causal chain, where each action causes only the action happening directly after it. This simple solution works well for MAKEBELIEVE, because this is the way they create stories. However, this is not how things would happen in the real world. Usually, an event can be caused by more than one event, or be caused by an event that happened in the past. The Virtual Storyteller simulates this kind of causality, so we need a way of storing stories that supports this.

An alternative that supports a causal network instead of a chain is the DPOCL story plan. DPOCL uses an approach from planning theory to save the actions that occur in a story to achieve specific goals. The result is similar to the Fabula model: a network graph that stores the events of the story. There are however differences. Because DPOCL is based on a plan structure, it sometimes has characters

perform actions just to achieve a main story goal. For example, in figure 3.3, Joe moves to the docks so that he can be captured by Sam. In the Virtual Storyteller characters could reason out of character to achieve the same effect, but there would also be an in-character reason for Joe to move to the docks, for example that he was hungry and there was a restaurant at the docks.

Because of its focus on a global plan, instead of characters, the DPOCL plan also does not contain the emotions of the characters, like the Fabula model has with the Internal Element. These differences make DPOCL less suitable for the character-based approach of the Virtual Storyteller.

The Fabula model uses the four causal relations from Trabasso et al. (1984): physical causality, psychological causality, motivation and enablement. This is not the only way of expressing causality. Another way is the classic model from Aristotle, which defines four types of causality (Falcon, 2012):

Material cause Something caused by the material of something, such as the bronze of a statue.

Formal cause Something caused by the form of something, such as the shape of a statue.

Efficient cause "The primary source of the change or rest", such as the artist casting the statue.

Final cause Something happening for the sake of some goal, e.g. the artist melting casting bronze *because* they want to make a statue.

Aristotle's model is based on teleological explanations of events, that is to say it is based on a natural or physical explanation of why things happen. This is opposed to Trabasso's model, which is based on the psychological reasoning of characters with beliefs, desires and intentions. As the Virtual Storyteller is based on agents with beliefs, desires and intentions, it makes more sense to use Trabasso's causes.

So far it looks like the Fabula model is the best way of expressing the events of the Virtual Storyteller. But that does not mean the current way of storing the Fabula graph is perfect. The current Fabula model is saved in the TriG format, which is a format made specifically for RDF. To make the Fabula model easier to import, a different format could be used. XML is a good choice, since it is standardized and used often. Expressing RDF in XML is possible by using a format such as TriX or RDF/XML (Worldwide Web Consortium, 2004). However, it's also possible to use something entirely different, as the new Virtual Storyteller does not have to use RDF.

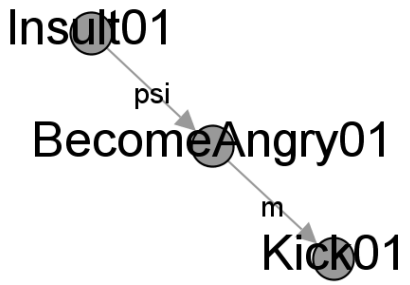

Figure 3.4: A minimal Fabula, as rendered by Gephi.

We held a meeting with all the people currently involved with the Virtual Storyteller to decide on a data format. We proposed that the Fabula structure will stay the same, but will be stored in the GraphML format. GraphML is a standardized format for graphs, and can be opened in existing graph editing software such as Gephi (Bastian et al., 2009). GraphML allows us to add information to each node and edge. For the nodes, this can be used to store information regarding a story event, such as the subject and object of an action. For the edges, it will be used to store the type of causality this edge represents, i.e. physical, psychological, motivation or enablement. A (minimal) example of a Fabula in GraphML format, as rendered by Gephi, can be found in Figure 3.4. This graph represents a story that involves one character insulting another character. This makes the second character angry, which results in him retaliating by kicking the first character. Example GraphML code for a node and edge of this story can be found in Figure 3.5. This figure contains a Fabula node about a police officer insulting a bystander. The edge then tells that this caused another event, which is called BecomeAngry01. This second node is not shown to keep the example simple.

3.3 Rhetorical Dependency Graph

The Rhetorical Dependency Graph is a format defined by Hielkema (2005) to serve as input for her Dutch surface realizer. There are two parts of the RDG: the Rhetorical Dependency Trees, which each describe a sentence, and the Rhetorical Dependency Graph itself, which contains the relations between sentences. I will describe both of them in this section.

```
<node id="Insult01">
  <data key="EventType">action</data>
  <data key="Type">insult</data>
  <data key="Agens">policeofficer</data>
  <data key="Patiens">bystander1</data>
  <data key="Time">1</data>
</node>

<edge id="Psychological01" source="Insult01" target="BecomeAngry01">
  <data key="RelType">psi-causes</data>
</edge>
```

Figure 3.5: Example GraphML code for a node and edge of the Fabula graph.

The Dependency Tree is a way of storing a sentence in a way that does not depend on word order, and is language independent. It is a tree structure that stores sentences by describing of each sentence part (such as the verb, or subject) its role in the sentence. It's based on the idea of Meaning Text Theory (Melčuk, 1988), that states information should not be stored based on a linear order, but instead as a syntactic relation, stored as labeled dependencies.

Using dependency trees for Natural Language Generation is not a new idea. A lot of surface realizers, such as RealPro (Lavoie and Rambow, 1997) and KPML (Bateman, 1997) use a dependency structure as input. The dependency trees used by Hielkema are based on the parse trees from Alpino, a system that can parse Dutch sentences (Van der Beek et al., 2002). Alpino's dependency trees are its output, while they serve as input for the surface realizer. This makes sense, as parsing is essentially the reverse of language generation.The Alpino trees were only slightly modified: a morphology tag was added so the surface realizer can inflect words.

Dependency trees start with a root node, to which the other sentence parts are connected. This root node can be the main verb, or an empty root node. The dependency trees used in the Narrator use the latter approach, where the main verb is instead a node in the sentence. To describe the role of each sentence part, a dependency label is used. These labels are based on the Corpus Gesproken Nederlands (Oostdijk, 2002). Figure 3.6 contains an example dependency tree for the phrase "Ik sla Adrie met een schoen" (I hit Adrie with a shoe).

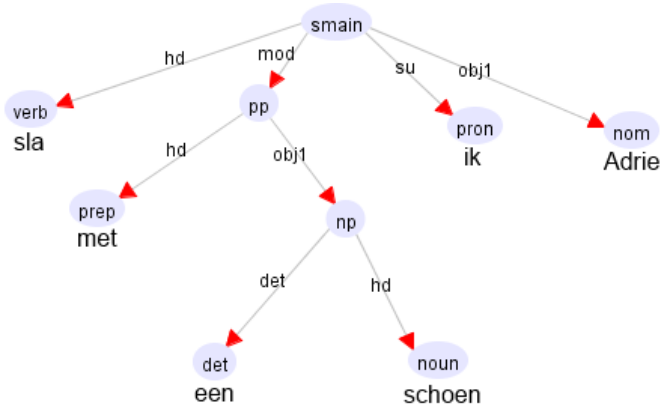

Figure 3.6: An example dependency tree for the sentence "Ik sla Adrie met een schoen".

The second part of the input to the surface realizer is the Rhetorical Dependency Graph (RDG). The RDG connects the sentences, which are stored as dependency trees, based on the rhetorical relation between them. Examples of such relations are causal, contrast or additive. The idea behind the RDG is based on Rhetorical Structure Theory (RST), a descriptive frame work on organizing natural text (Mann and Thompson, 1987).

Knowing the relation between clauses is important for the task of aggregation, i.e. the combination of two sentences into one. Based on the relation between them, sentences are combined differently. Take the sentences "Barry is scared" and "The police officer is happy". If they were connected by a causal relation, the sentence would be something like "The police officer is happy *because* Barry is scared.". An additive relation would result in "Barry is scared *and* the police officer is happy", while a contrast relation would be "Barry is scared *but* the police officer is happy".

In the example sentence I've highlighted the words *because*, *and* and *but*. These words are called cue words, and they are a big part of how the surface realizer realizes rhetorical relations. Figure 3.7 contains a combination of the clauses "Hij beledigt hem" (he insults him) and "Barry is boos" (Barry is angry) using a cause relation. In this story, the police officer has insulted the adolescent Barry, which causes Barry to be angry. The surface sentence generated was "Hij beledigt hem, dus Barry is boos."

(He insults him, therefore Barry is angry).

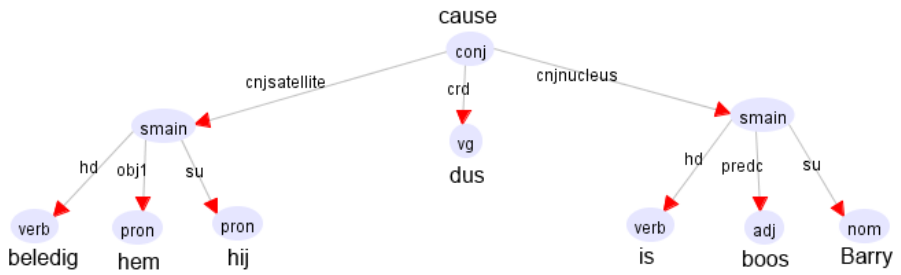

Figure 3.7: An example combination of the clauses “Hij beledigt hem” (he insults him) and “Barry is boos” (Barry is angry) using a cause relation.

The RDG used by the Narrator supports the following rhetorical relations: cause, purpose, contrast, temporal, additive and relative.

3.4 Old Narrator

This section describes the existing Narrator system for the Virtual Storyteller. It is based on Nanda Slabbers’ Master thesis (Slabbers, 2006) and the system description by René Zeeders (Zeeders, 2008).

The Narrator consists of three parts: the *Document Planner*, the *Microplanner*, and the *Surface Realizer*. I will discuss each of the three modules. A more detailed description of the Narrator System, including my modifications, can be found in Chapter 7.

Throughout this description I will use an example story from Slabbers’ thesis as an illustration to the tasks of the different steps.

3.4.1 Fabula

As described in Section 3.2, the Fabula is the input to the Narrator. For this example, let’s start with a Fabula containing a story about a knight and a princess, as found in Figure 3.8. The plot elements of the story are given as simple sentences in Figure 3.9. Normally the Fabula would be generated by the Virtual Storyteller, however in this case it was manually written by Nanda Slabbers.

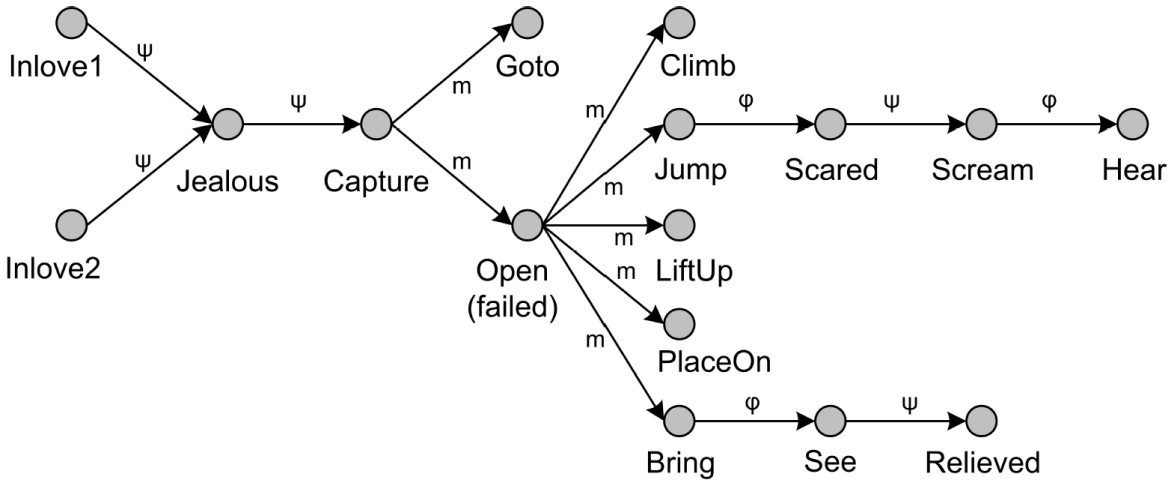

Figure 3.8: An example Fabula graph for a story about a knight and a princess (Slabbers, 2006, Figure 9.1).

3.4.2 Document Planner

The responsibility of the Document Planner is to define the structure of the story. It does this by creating a Document Plan, which serves as input for the Microplanner.

The InitialDocplanBuilder is responsible for converting the Fabula graph to the document plan, a dependency tree structure containing the story events as leafs, connected by rhetorical relations. This

InIove1 A knight was in love with a princess.

InIove2 The princess was in love with a prince.

Jealous The knight was jealous.

Capture The knight wanted to capture the princess.

Goto The knight went to a castle.

Open The knight tried to open a gate.

Climb The knight climbed in a tree.

Jump The knight jumped into the princesss bedroom.

Scared The princess was scared.

Scream The princess screamed.

Hear Nobody heard the princess.

LiftUp The knight lifted up the princess.

PlaceOn The knight placed the princess on his horse.

Bring The knight brought her to a bridge.

See The princess saw the prince.

Relieved The princess was relieved.

Figure 3.9: The plot elements for the Fabula in Figure 3.8, informally described as English sentences.

structure is similar to the Rhetorical Dependency Graph as discussed in Section 3.3, the difference being that the document plan does not contain sentence plans. Instead, the story events are stored as plot elements, which are more high-level descriptions of the events in the story.

The document plan is created by first reading the Fabula, and converting each Fabula node into a plot element in the dependency tree. The causal relations between events are then converted to the correct rhetorical relations. This module uses a chronological ordering for story events.

After creating an Initial Document Plan, the Document Planner performs three transformations on the created dependency tree. The first is called Background Information Supplier (BGIS). The BGIS adds information from the story world and information about the characters to supply the reader with background information. For example, the BGIS will add a character's name to the story to introduce them the first time they are mentioned.

The second transformation is done by the State Transformer. This part decides how plot elements which correspond to an Internal Element of a character can be described best. It can describe the Internal Element as a separate sentence ('the princess was scared'), but it can also describe the element as an adjective or relative clause. Finally, it can use a piece of canned text to describe the internal element as an action ('her heart was pounding from fear'). The third transformation is the Mood Creator. This part adds canned text describing the story world, based on the 'mood' of the story, which is in turn based on the emotions of the characters. The Mood Creator is currently not used, as it is difficult to know the emotional state of the characters at each time point.

The Branch Remover performs the final transformation of the document plan. This part tries to change the document plan so that it doesn't contain long branches which cannot be combined by the surface realizer. This would result in simple, short sentences, so making these branches shorter would result in better surface text.

Figure 3.10 gives an example document plan for the knight story. The document plan was manually split into paragraphs, as this is not implemented in the old Narrator.

3.4.3 Microplanner

This part uses the document plan to create the sentence structures of the story. This is done by taking each story event in the plan created by the Document Planner, and converting it to a sentence. The Microplanner does this by using a template for each kind of plot element (actions and events, internal states, perceptions and beliefs, goals, and settings).

The sentences are created as dependency trees containing information about the story event: what

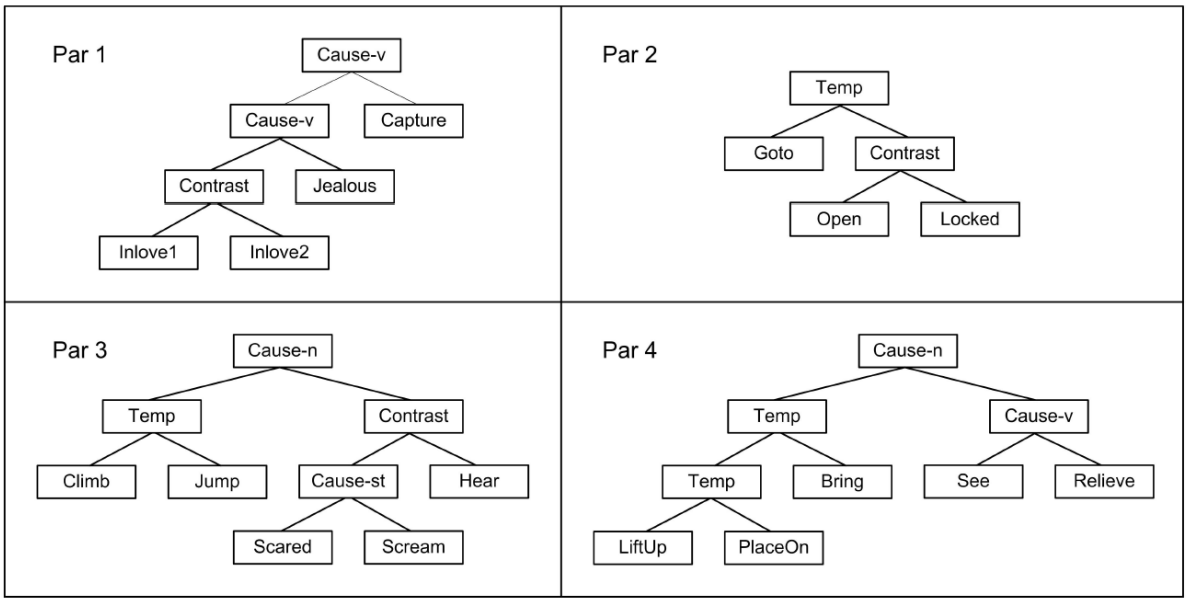

Figure 3.10: The (manually created) document plan for the knight story (Slabbers, 2006, Figure 9.4).

characters and objects were involved, what action was performed, and what the structure of the sentence should be. The dependency trees do not contain information on the ordering of the words within the sentence.

After filling the templates of the dependency tree the Microplanner performs lexicalization. The Narrator has a lexicon for this, which maps the concepts, such as characters, objects and actions, to actual words. It also stores the previous words used for each concept, so the Microplanner can choose a different word for a specific concept if several words were specified in the lexicon.

An example sentence plan from the knight story can be found in Figure 3.11.

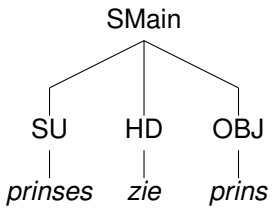

Figure 3.11: An example sentence plan for the sentence ‘de prinses ziet de prins’ (the princess sees the prince).

3.4.4 Surface Realizer

The final step is carried out by the surface realizer, which converts the sentence plan generated into surface text. It’s based on the surface realizer made by Hielkema (2005). First, it transforms the rhetoric relations in the dependency tree into their final form. It then processes the Rhetorical Dependency Graph depth-first, and tries to combine the dependency trees based on rhetorical relations between them. It has a conjunctor for doing the actual combining, and an elliptor for syntactic aggregation.

The second step is creating referring expressions for each entity, using the algorithm from Krahmer and Theune (2002). The last component is the surface form generator, which takes the final tree that was combined by the first step, and transforms it into the final text. It uses the referring expressions created in the previous step, decides the final order of sentences, inflects words and adds punctuation.

The final output of the story can be found in Figure 3.12.

3.5 Extensions and modifications

In this section I will look at what other changes could be made to the Narrator, apart from focalization and flashbacks. The biggest changes come from changing the domain from fairy tales to police training, but there are some smaller things that could be changed as well.

1. Er was eens een buitengewoon mooie prinses, die Amalia heette. Een ridder van een ver land was verliefd op haar, maar zij was verliefd op een jonge prins. De ridder was jaloers, dus hij wilde haar ontvoeren.

2. De prinses, een lief meisje, woonde in een groot kasteel. Op een nacht ging de ridder naar het kasteel. Voorzichtig probeerde hij de zware poort te openen, maar die was op slot.

3. Nadat de ridder in een hoge boom was geklommen, sprong hij de slaapkamer van de prinses binnen. Zij was zo geschrokken, dat zij hard schreeuwde, maar niemand hoorde haar.

4. Hardhandig pakte de ridder de tegenstribbelende prinses op en vervolgens zette hij haar op zijn paard. Daarna bracht hij haar naar een oude en smalle brug. Aan de overkant zag zij de prins, op wie zij verliefd was. Wat was prinses Amalia opgelucht!

Figure 3.12: The final output of the example story about a knight and a princess (Slabbers, 2006, Figure 9.5).

3.5.1 New domain

Data format As both the Virtual Storyteller and the Narrator are updated, this is the perfect time to think about a different data format for communication between the two. The following data structures could be updated for communication between the Virtual Storyteller and the Narrator: the Fabula model that contains the story, the lexicon, and the format for storing the story world and characters.

In Chapter 6 I will discuss the changes that were made to the data formats.

Domain-specific language generation The old Narrator contains some parts that are made with the fairytale domain in mind. For example, the old Narrator has an option of starting a story with 'er was eens' (*once upon a time*) and specific pieces of information about entities in the fairytale domain, such as the princess, knight and castle, are hard-coded. These parts should be changed so that they work with any domain, for example by specifying domain-specific text in an external file. It's a good idea to not just make the Narrator for the new police domain, but also make it easy to use it for another domain.

Connection with AGENT The new Narrator should not just be compatible with the serious game through exchangeable data formats, it should also be able to connect to a simulation and narrate the events in it. To achieve this, a program should be made that connects to a server running the simulation. It should receive the events from the simulation, as well as information on the story world and characters. The Narrator should then use this information to create a report in natural language.

3.5.2 Other modifications

Apart from the bigger changes mentioned before, i.e. the new domain, focalization and flashbacks, there are a few small things than can be changed, which I will discuss here.

User interface When testing the old Narrator, I found the user interface to be a bit confusing. It does not look very native to the rest of the operating system: the buttons, input fields, layout and text font are different compared to the rest of Windows, which makes the user experience inconsistent with other programs. More importantly, there are a lot of file input fields for which it can be unclear what file should be chosen. This makes it very hard to use the Narrator, as it requires the user to know exactly which file belongs in which field.

I included a screen shot of the old Narrator in Figure 3.13. To make testing and reuse by future users easier, the new Narrator should include a GUI that is easier to use and looks cleaner. The file input format should also be changed, so it's not required to manually specify all files which are required by the Narrator. Instead, a single input file could be used which refers to the files it requires.

Character introduction The story could start by introducing the characters, by telling their name and location. The Background Information Supplier already tells the name of the character when it is first mentioned, but this is done in the normal text, as opposed to an introduction at the start, and does not mention the location of the character.

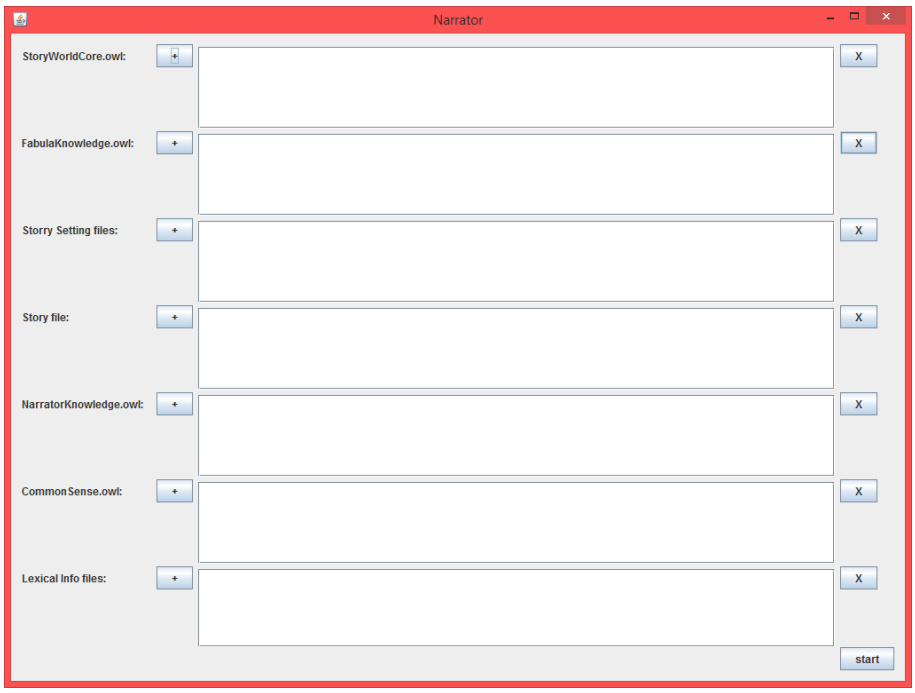

Figure 3.13: The Graphical User Interface from the old Narrator, in Windows 8.1.

Chapter 4

Focalization

4.1 Introduction

Focalization refers to *"a selection or restriction of narrative information in relation to the experience and knowledge of the narrator, the characters or other, more hypothetical entities in the storyworld"* (Niederhoff, 2011). The term itself was coined by Genette and Lewin (1983).

Genette defined three types of focalization: zero, internal and external focalization. In stories that have zero focalization the perspective is that of an all-knowing narrator. This is how the Narrator currently functions. The story is told using all available information, without using the perspective of a certain character. Internal focalization refers to the story being told by a character in the story itself, while external focalization refers to the perspective from someone outside the story. The difference between zero and external focalization is that the storyteller in external focalization isn't all knowing.

Having internal focalization, i.e. having a character in the story that tells the story, allows the writer to use other narrative modes. In a first person perspective, the focalized character tells the story as it happens to them, using *I* to refer to themselves. This seems most fit for when you want to put the reader in the shoes of the focalized character. In the serious game, this could be used to tell the story from the loitering youth's perspective, to help put the training police officer in the shoes of the youth.

The second person perspective is not often used in literature, but it could be used to tell the actions of the police officer, as a direct report from the interaction in the serious game. (Schofield, 1998)

The last option is third-person narrative. This looks the least different from an all-knowing narrator. In literature, the difference is usually that this represents the story from a certain character's subjective view. In the case of external focalization, this character does not participate in the story. In the Narrator, the only difference is that a focalized third person story only contains events that the focalized character can see or participates in.

4.2 Related Work

Curveship In Section 2.2 I described Curveship, an interactive fiction framework created by Montfort (2011). To achieve focalization, Curveship makes a distinction between the 'actual world' and the 'concept', the world as perceived and believed by each actor. Using this technique, Curveship can either tell the story from the point of an all-knowing narrator, or from the point of view of one of the actors, by creating the appropriate concept.

Gervás Gervás (2013) describes a method of using focalization for content selection. He uses the events in the log of a chess game as an example. The game is focalized for each chess piece, to create a sequence of events that this piece takes part in, or can perceive. For perception it is given that a piece can observe all actions in an NxN square around it. This focalized sequence of events is called a *fiber*. The system then performs a task called *heckling*, which involves tracking perception, tracking the characters (or pieces) in the game events, and actually constructing a fiber for each character.

To create a narrative out of individual fibers, the system uses an evolutionary algorithm to select a set of fibers according to certain criteria: overlap, how many events are told more than once, coverage, how much of all events is present in this story, density, how many events are captures instead of just moves, cohesion, the ratio between total moves covered and moves covered by the longest fiber, and number of focalizers, which should ideally be between 2 and 6. In the example chess game, the optimal fiber set turned out to be the combination of the fibers of a black pawn and a white bishop.

4.3 Design

Before designing an algorithm for focalization, I defined focalization as that the story only contains events that the focalizing character can either perceive, or that they themselves participated in. The algorithm that deals with focalization should therefore remove all Fabula nodes that don't fit these criteria. Fabula edges that belong to a removed node should also be removed, to prevent the graph having edges that lead to nowhere.

The algorithm used to transform the full graph into a focalized graph can be found in Figure 4.1. Note that the algorithm does not differentiate between events that the character participated in, or events that they perceived. In the Fabula, perceptions are present as story elements containing the perceiving character as agens, so using this algorithm perceptions of the focalized character will be correctly added.

```
1 for node in graph
    if !containsCharacter(node)
3       removeNodes(node)
        removeEdges(inEdges(node))
5
  containsCharacter:
7   character = FocalizedCharacter
    return node.agens == character | node.patiens == character
9       | node.target == character
```

Figure 4.1: The algorithm to transform a graph into a focalized graph in pseudocode.

```
Een politieman, die Adrie heet, is in een steegje.
Hij wil met een stoere hangjongere, die Barry heet, praten en hij
gaat naar een hoofdstraat.
Hij beledigt hem, dus Barry is boos.
Met een schoen slaat hij de politieman, zodat hij verdrietig is.
Een omstander, die Manuela heet, gaat naar de hoofdstraat en zij
ziet dat de hangjongere hem slaat.
```

Figure 4.2: A basic, unfocalized story about a cop, a loitering adolescent and a bystander.

As an example, take the automatically generated unfocalized story in Figure 4.2[1] [2] We want to tell this story through the perspective of Barry, the loitering adolescent. To do this, the Narrator filters the story using the rules in Figure 4.1, then proceeds as normal. This results in the story of Figure 4.3[3], a focalized third person perspective story.

```
Een politieman beledigt een stoere hangjongere, die Barry heet.
Hij heet Adrie.
Daarom is de hangjongere boos.
Met een schoen slaat hij de politieman.
```

Figure 4.3: The story in figure 4.2, focalized from the perspective of Barry, the loitering adolescent.

Adding perspective to this story is fairly straight-forward. The Referring Expressions Generator needs to be changed to always return 'I' for the focalized character in first person perspective, or 'you' in second person perspective. The Sentence Planner and Surface Realizer also need to be changed to correctly inflect verbs. This is done by changing the 'tag' the Sentence Planner adds to a verb when the focalized character is the subject of the sentence to include first or second person, depending on perspective. This tag then tells the Surface Realizer how to inflect this verb. Adding these changes results in the story of Figure 4.4[4].

[1] Note that all stories in this chapter are automatically generated and contain some language generation errors. In this example. 'een hoofdstraat' should be 'de hoofdstraat'. The Narrator currently has no way of knowing whether a location (or other concept) is unique, so it assumes each concept is not unique and uses an indefinite article. Referring expression generation also does not work like it should.

[2] Translated: A policeman called Adrie is in an alley. He wants to talk to a tough loitering youth called Barry, and he goes to the main street. Hij insults him, therefore Barry is angry. He hits the police officer with a shoe, so he is sad. A bystander called Manuela goes to the main street and she sees that the loitering youth hits him.

[3] Translated: A policeman insults a tough loitering youth, called Barry. He is called Adrie. Therefore the loitering youth is angry. He hits the police officer with a shoe.

[4] Translated: A police officer insults me, who is called Barry. He is called Adrie. Therefore I am angry. I hit the police officer with a shoe.

```
Een politieman beledigt mij, die Barry heet.
Hij heet Adrie.
Daarom ben ik boos.
Met een schoen sla ik de politieman.
```

Figure 4.4: The focalized story from Figure 4.3, in first person perspective.

As can be seen in Figure 4.4, the way the Narrator adds names doesn't work well in this perspective. Therefore, I changed the module that tells the name of each character to not mention the name of the focalized character if the story is being told in first or second person perspective. This results in the final story of Figure 4.5[5].

```
Een politieman, die Adrie heet, beledigt mij, dus ik ben boos.
Met een schoen sla ik hem.
```

Figure 4.5: The focalized story from Figure 4.4, changed to not mention the name of the focalized character.

Reduced form of the second person pronoun

When I first added focalization to the Narrator, it always chose the full form for the second person pronoun ('jij' or 'jou'). However, Dutch also has a reduced form: 'je'. While it is grammatically correct to use the full form, it sounds unnatural to always do so (Geerts et al., 1984, Paragraph 5.2.7). In general, the rule is that the full form should be used when the stress is on the entitiy to which the pronoun applies. This is for example the case in contrast relations, or when you're directly addressing someone. Otherwise, the reduced form should be used.

In the case of the Narrator, the second person pronoun is only used in the context of focalization. The report generated by the Narrator tells the reader what they have done in a certain story. As the Narrator is therefore always addressing the reader directly, it makes sense to use the full form. However, this sounds unnatural after a few times.

It is hard to find examples of literature written in the second person perspective, as it is not often used in literature. One example is is used by Altman (1984) in the prologue of a horror novel. The second person perspective is here used to put the reader in the shoes of a pregnant woman who is about to kill her unborn child. In this section, the translator only uses the reduced form, presumably to pull the reader even closer to the character.

For now I chose to always used the reduced form instead of the full form. This is not a perfect solution, as a stressed pronoun could be needed in some case. Take for example the following sentence with a contrast relation: 'Ik ga naar huis, maar jij blijft hier' (I'm going home, but *you* are staying here). In this case, only the full form is correct. Currently, the Narrator never creates contrast relations, however newer versions of the Narrator that do should take care to use the full form of the second person pronoun in these cases.

[5]Translated: A police officer called Adrie insults me, so I am angry. I hit him with a shoe.

Chapter 5

Flashbacks

5.1 Introduction

The simulation program for training police officers can be run multiple times with the same set of characters. The actions of the trainee in the previous sessions will then influence the behavior of the A.I.-controlled NPCs in the current session. It is of great value to the learning experience that the trainee knows the causes behind A.I. behavior (Core et al., 2006). Therefore the Narrator should describe what events from previous sessions caused the characters to behave a certain way. That is what flashbacks do: they link a past event to a current situation.

Flashbacks are a form of anachrony, which is defined as a difference between the order in which events are told in the story and in which they occurred (Prince, 2003). Besides flashbacks, there are other forms of anachrony. One such form is the flash forward in which the story tells something from the future, for example to hint at an event that is going to happen. Another example is retrograde, in which a part of the story is told in the opposite order of which the events occurred.

Prince (2003) defines a flashback (or analepsis) as an anachrony that goes back to the past, compared to the current story time, or the telling of one or more events that happened before the current story time. He also defines two properties of analepsis: the *reach* and *extent*. The reach is how long before the current events the events of the flashback took place. The extent is the total duration of the events in the flashback. For example, for this flashback 'Barry is angry because he had an hour-long fight with his dad two days earlier', the reach is two days and the extent is one hour.

Genette and Lewin (1983) have looked at the temporal order of narrative. They did this by studying the duality of story time and narrative time. In the history of storytelling, folklore stories were often chronological. However, when looking at classical literature Genette found examples of anachrony. Already at the start of the *Iliad* he found an order that closely resembled a retrograde [1]. Analepsis can also be found in Homer. An example can be found in the *Odyssey* where Odysseus in disguise is recognized by his old nanny Eyrucleia, when she sees a scar on him. The flashback consists of telling the moment Odysseus got the scar, and is used to explain to the reader (or listener) how Eyruclea recognized him [2].

5.2 Related work

Flashbacks are not typical for natural language generation systems. Gervás et al. (2006) mention that in their survey research, they have not found a story generation system at the time which uses flashbacks. Despite this, I've been able to find a few systems which support anachrony, which I will describe here. Note that these were all created after the survey research by Gervás et al.

Curveship

Montfort (2007) describes how Curveship handles achronological ordering of story events. This paper describes algorithms in pseudocode which can be used to order a part of story in different ways: chronological, retrograde, zigzag, analepsis (flashback), prolepsis (flash forward), syllepsis (category ordering) and achrony, which he defines as randomly ordering events, are supported.

Analepsis or flashbacks are defined as inserting an anachronism into a (chronological) sequence of events. To do this, Montfort wrote an algorithm, which I included in Figure 5.1. The input to this algorithm is as followes: S, the main sequence of events, A, the set of all events, R, a rule for selecting events for a flashback, and n, the number of events before analepsis occurs, i.e. the point of insertion. It

[1] Homer, *Iliad* 1.1-32
[2] Homer, *Odyssee* 19.361-475

works by adding events from the main sequence to the new main sequence until the point of insertion is reached. Then, it adds the flashback event following some heuristic rule, a few examples of such rules are mentioned later. Afterwards, it adds the rest of the main sequence. The result is the main sequence with one or more flashbacks added.

```
U←()
while |U| < n: append(U, pop(S))
append(U, R(A))
append(U, S)
return U
```

Figure 5.1: Montfort (2007)'s algorithm for analepsis.

According to Montfort, the reach and extent are not very interesting to define when writing flashbacks in interactive fiction. A more interesting challenge is to find *which* previous event(s) should be inserted, and *when*. It's still difficult to know when a deviation from the normal chronological order makes the story better. To do this, Montfort recommends using simple rules as heuristics. He describes two heuristics in this paper: "Select the most salient event from the first time the focalizer encountered this character" and "Select the most salient things that the focalizer has seen happen in this room in the past, up to three of them." (Montfort, 2007, Page 90)

Bae & Young Bae and Young (2008) describe a system that can generate both analepsis (flashbacks) and prolepsis (foreshadowing) with the specific goal of evoking surprise in the reader.

Their approach uses a generate and test method, starting with a chronological fabula generated by DPOCL (Young and Moore, 1994). This fabula is then given to a their generator, which selects events that can be used as flashbacks. This generator starts by identifying significant events, by selecting events that directly connect to a goal event. It then chooses one of the events causing this significant event, called initiating events, and makes it a flashback by telling it after the event occurred. This is based on the idea that the absence of some initiating events can cause curiosity or surprise.

The generated flashbacks are then evaluated, based on their unexpectedness. The evaluator searches for a plan that can satisfy the preconditions of the significant event without the flashback. I.e., the significant event can be told completely without needing the flashback. In this case, the evaluator determines that the flashback is expected and not surprising, and the generator creates another one.

5.3 Design

To illustrate the way I created flashbacks, I will give an example of flashback that is not directly used to explain a story event but instead explains the emotional relationship between two characters. Pacific Rim is a 2013 science fiction movie about people that pilot robots to fight monsters. In this movie, pilot Mako Mori is shown as being very loyal to her commander Pentecost, much to the chagrin of fellow pilot Raleigh Becket. Halfway during the movie, a scene is shown that took place in the past, in which Pentecost saves Mori (del Toro and Beacham, 2013, 51m50s, continued at 1h01m10s). By showing this scene, the viewer is supposed to better understand Mori's behavior towards Pentecost.

Creating elaborate flashbacks like the one in Pacific Rim will be very hard. What I instead did is use it as inspiration to make flashbacks based on emotions, while following Montfort (2007)'s idea of using a simple heuristic rule. The idea is as follows: if a certain character has an emotion that is not caused by an event in the current story, it must have been caused by something in the past. By looking at a story file with all previous events, a flashback module could find the latest event that caused a certain emotion for a character. This cause can then be added to the current story.

Figure 5.2 describes the first part of the flashback algorithm: finding the causes of emotions from the graph with previous events. Previous events are stored in the same way as the current story, by means of a Fabula file (see Section 3.2) containing a graph of story events connected by causes. If the algorithm encounters a cause of a certain emotion for a certain character, it stores this information. If it encounters the same combination of emotion and character, it checks to see if the second one it has encountered is newer than what's already stored, and replaces the old one if it is.

Figure 5.3 describes the second part of the algorithm. This part takes the current Fabula graph and checks to see if there are any emotions without a cause. If that is the case, it looks in the stored causes and adds a cause node if one is applicable to this combination of character and emotion. To prevent adding the same cause twice and creating a repetitive story, the algorithm stores which nodes have already been added.

Each node that is added is marked as being a flashback. This is later used by the sentence planner, which adds information to the sentence plan so that flashback events are narrated in the perfect tense.

```
for node in backgroundGraph
    if n.type == ”state”
        for edge in inEdges(node)
            if edge.type == ”psi−causes” || edge.type == ”phi−causes”
                causeNode = getSource(edge)
                tuple = (node.agens, node.emotion)

                //check if this is actually the latest cause for this
                //emotion. if the map contains an earlier one,
                //replace it with this one
                previousNode = emotionCauses(tuple)
                if (previous==null)
                    //if no other exists, add it without problems
                    emotionCauses.put(tuple, causeNode)
                else
                    //if another exists, replace only if the other one is
                    //from an earlier time
                    if previous.time<=causeNode.time
                        emotionCauses.put(tuple, causeNode)
```

Figure 5.2: The algorithm to read the previous causes of emotions, in pseudocode.

```
for node in graph
    //If the node is an emotion
    if node.type == ”state”
        //And does not have incoming edges that
        //directly cause it (psi−causes or phi−causes)
        add = true
        for edge : inEdges(node)
            if edge.type == ”psi−causes” || edge.type == ”phi−causes”
                add = false

        //add a cause if it exists.
        if add
            test = (node.agens, node.name)
            cause = emotionCauses.get(test)

            //if already added, don’t add it again.
            if p != null && !alreadyAdded(cause)
                setAlreadyAdded(cause)
                //the cause should have the timestamp of the effect, minus 1
                cause.time = node.time − 1
                cause.isFlashback = true
                addNode(cause)
                addEdge(new edge(”psi−causes”,cause,node))
```

Figure 5.3: The algorithm to add flashbacks to the current story graph, in pseudocode.

An example story using a flashback can be found in Figure 5.4[3] [4]. In this story, the loitering adolescent called Barry is still mad from being insulted by the police officer at a previous encounter. This causes him to yell at the police officer.

```
Een politieman, die Adrie heet, gaat naar een hoofdstraat.
Een stoere hangjongere, die Barry heet, ziet dat de politieman in
de hoofdstraat is.
Eerder heeft hij hem beledigd, dus de hangjongere is boos en hij
schreeuwt naar de politieman.
```

Figure 5.4: A simple story with flashbacks.

[3] Translated: A police officer called Adrie goes to a main street. A tough loitering youth called Barry sees that the police officer is in the main street. Earlier he has insulted him, therefore the loitering youth is angry and he yells at the police officer.

[4] As with the stories in Chapter 4, there are still some language generation errors in this story.

Chapter 6

Data Representation

6.1 Introduction

To write text we first need to know something about what we need to write. As the goal is to write reports about the events from a session in a serious game, this game should pass these events, as well as the information on the world and background of the story, over to the Narrator. To make sure this goes well, we need to define data formats for every piece of information the Narrator needs.

In Sections 3.2 and 3.3 I already described the Fabula Model and the Rhetorical Dependency Graph, existing formats that stay essentially the same for the new Narrator. In this chapter I will focus on two new data formats: the lexicon and the scenario file.

Section 6.2 describes the lexicon, the data format that is used to store information about all the words that can be used in the story. This includes grammatical information such as verb tenses.

Finally, Section 6.3 describes how the Narrator should store information about the characters, items, locations and other entities in the story world. The characters will be described using an example loitering youth character.

6.2 Lexicon

6.2.1 Introduction

In this section I describe the lexicon, the data structure that defines the words that can be used in the stories the Narrator generates, including grammatical information, such as verb inflection. The Narrator's lexicon was designed in three steps, as presented in this section. First, I looked at basic literature on Dutch grammar to find the rules for inflection of Dutch words. Next I look at how the lexicons for other NLG systems were designed. Finally, I use this information to design my own lexicon for the Narrator.

6.2.2 Analysis of Dutch language

In this section I look at the Algemene Nederlandse Spraakkunst (ANS) (Geerts et al., 1984), a book that defines the grammatical rules for Dutch. To find out what information the lexicon should store, I specifically look at the rules for inflections of adjectives, nouns and verbs.

Adjectives

Adjectives are not very complicated in Dutch. They have only one inflected form which is usually made by adding an 'e' to the end of the adjective. As this rule applies to the pronunciation of words and not the spelling, it is not always as simple as using the root +'e' in written text. For example, adjectives that have a double vowel in the root lose that vowel in the spelling of the inflected form, for example *groot* becomes *grote*. Other adjectives change their final consonant, e.g. *lief* becomes *lieve*. There are also adjectives that don't have a difference between the root and the inflected form, such as *lila* or *sexy*.

The inflected form is used when the adjective is preceded by a definite article (de or het). It is also used when the noun to which the adjective applies is male or female, i.e. not neuter, and an indefinite article (een) is used (Geerts et al., 1984, Paragraph 6.4.1.2).

Nouns

Like adjectives, Dutch nouns have only one inflected form. This is the plural form, which is usually made by adding -s or -en to the root (Geerts et al., 1984, Paragraph 3.5.1). Even though this sounds easy

enough, it's difficult to know which suffix should be used, and there are many exceptions. Even words that look very similar can have different ways of forming the plural. For example, the word *loopgraaf* (trench) is *loopgraven* in the plural form, but the word *fotograaf* (photographer) becomes *fotografen* (Geerts et al., 1984, Paragraph 3.5.2.2).

Nouns also have a grammatical gender (Geerts et al., 1984, 3.3). Dutch has three genders: male, female or neuter. Male and female words use *de* as the definite article, neuter words use *het*. Some words can be both male and female, if they refer to a person. The word changes its gender to that of the person it's referring to. In the Dutch that is spoken in the Netherlands there is no difference between how male and female objects are referred to, they are normally referred to as *hij* (he) no matter the actual grammatical gender of the word. In the Flemish dialect however, this is different, as Flemish people will often use *ze* (she) to refer to an object that is grammatically female (Geerts et al., 1984, Paragraph 3.3.3).

To conclude, to correctly inflect nouns, and to correctly inflect the adjectives and articles that belong to them, we need the following information in the lexicon:

- Root
- Plural
- Grammatical gender

Verbs

It's more complicated to create a surface form for Dutch verbs than for other types of words, as a single verb can have a lot of different conjugations, which all result in different surface forms. While there are regular rules for conjugating verbs, a lot of Dutch verbs are irregular and do not follow these rules. Verbs also play a central role in Dutch sentences. Often the structure of the sentence, e.g. what objects are present, depends on the verb.

In the ANS I found a number of verb properties that could be added to the lexicon, as they could be useful for properly conjugating verbs and creating good sentences. These are the following:

- Root
- Conjugations
- Auxiliary verb
- Type of verb
- Preposition
- Preposition target
- Particle.

Root The root is the base of the verb, from which the rest of the verb is conjugated. In Dutch, the base is formed by removing 'en' or 'n' from the infinitive form (Geerts et al., 1984, Paragraph 2.3.2.3). As with adjective inflection, this rule applies to the pronunciation and not the spelling of words, so if the verb contains a single vowel that is pronounced as a double vowel, e.g. maken, the root (maak) has a double vowel. In Dutch, the root is usually the same as the singular first person present form.

Conjugations Verbs in Dutch have different surface forms for different configurations of person (first, second, third), number (singular/plural), and tense (present, past). To properly conjugate irregular verbs we need to know the first person singular present, second person singular present, third person singular present, plural present, singular simple past, plural simple past, infinitive, and the participle for the perfect and pluperfect forms (Geerts et al., 1984, Paragraph 2.3.2.8). A sample conjugation for the verb *zijn* (to be) can be found in Table 6.1.

1sg pres	2sg pres	3sg pres	Pl pres	Sg past	Pl past	Inf	Participle
ben	bent	is	zijn	was	waren	zijn	geweest

Table 6.1: The conjugations for the verb *zijn* (to be).

Auxiliary verb Dutch uses an auxiliary verb together with the participle to create the perfect and pluperfect forms. This verb can be either *zijn* (to be) or *hebben* (to have). To correctly create the perfect and pluperfect forms, the Narrator has to know which auxiliary verb is used.

Type of verb There are different types of verb, for example based on what objects a verb can have. There are intransitive, transitive and ditransitive verbs, that can respectively have no, one or two objects.

Preposition For some verbs the sentence planner has to add a preposition to a certain object. For example, the verb *praten* (to talk) needs the preposition *met* (with) if we want to specify who the subject is talking to. To properly add this preposition the Narrator needs to know which preposition should be added, if any.

Preposition target Additionally, the Narrator also needs to know which target the preposition applies to. For ditransitive verbs, the preposition can apply to a secondary object (I give the book *to my supervisor*), while for other verbs it can apply to a location (I walk *to the station*). To make sure the Narrator puts the preposition in the right place in the sentence, it is therefore important to store to which element in the sentence it applies.

Particle Some Dutch verbs are split in two parts: the root and the particle, which is split from the rest of the verb. For example, the verb oppakken (to pick up) becomes 'ik pak iets op' (I pick something up), where 'op' is the particle.

6.2.3 Related work on lexicons

Old Narrator

The format used by the old Narrator (Slabbers, 2006, Paragraph 7.3.1) has the following properties for all words in the lexicon:

- Entity
- Root
- Part of speech tag

For nouns, the following properties are added:

- Determiner
- Gender

The follow properties are added for verbs:

- Preposition
- Dependency label of the target node
- Particle

The conjugation of verbs is present in a separate 'verb lexicon', which is in the form of a text file called *verbs.txt*. If a verb is found in the normal lexicon but not in *verbs.txt*, the Surface Realizer assumes the verb is conjugated using standard rules for the Dutch language (Hielkema, 2005). The same happens for nouns: if a noun has a plural form that can't be formed using the standard rules it's included in a file *nouns.txt*.

The auxiliary verbs are stored in the code: there's a list of verbs that use *zijn* (to be) as an auxiliary verb, the old Narrator assumes that all other verbs use *hebben* (to have).

The dependency label specifies the label of the node in the dependency tree that is the target of the preposition (Slabbers, 2006, Paragraph 7.3.1). The preposition always applies to the Target of a Fabula node (see Section 3.2). The label specifies what function this Target node has in the sentence, for example secondary object or location. The dependency labels are based on the Corpus Gesproken Nederlands (Hoekstra et al., 2003) (see Figure 6.1) and match the Rhetorical Dependency Tree (see section 3.3). The Narrator assumes the Instrument node always has the preposition *met* (with), this does not have to be specified. This allows the Narrator to create sentences with two prepositions, such as the sentence *Barry gaat met de brommer naar het plein* (Barry goes to the square on his moped). In this case, the moped is the instrument, the square the target, and the verb specifies the preposition *naar* (to) with the dependency label *ld*, which specifies a locational component.

SimpleNLG

SimpleNLG is a surface realizer for English text that is designed to be as simple in use as possible (Gatt and Reiter, 2009). This simple design allows the client application to have more control over the generated surface text. Although SimpleNLG currently does not have an implementation for the Dutch language, the Dutch and English grammar are similar enough to learn something from how SimpleNLG handles lexicons.

SimpleNLG has a default lexicon for English, which is stored in the file default-lexicon.xml. This default lexicon is fairly small, so it also includes a lexicon called the "NIH Specialist lexicon", which

	d-LABEL	OMSCHRIJVING
hoofd	HD	werkwoordelijk hoofd (finiet of niet-finiet)
complementatie	SU	subject, onderwerp
	SUP	voorlopig subject
	OBJ1	direct object, lijdend voorwerp
	POBJ1	voorlopig direct object
	SE	verplicht reflexief object
	OBJ2	secundair object (meewerkend, belanghebbend, ondervindend)
	PREDC	predicatief complement
	VC	verbaal complement, werkwoordelijk deel van gezegde
	PC	voorzetselvoorwerp
	ME	maat (duur, gewicht, ...) complement
	LD	locatief of directioneel complement
	SVP	scheidbaar deel van werkwoord
modificatie	MOD	bijwoordelijke bepaling
	PREDM	bepaling van gesteldheid 'tijdens de handeling'

Figure 6.1: The verbal dependency labels from the CGN, as used in the Narrator (Hoekstra et al., 2003, Figure 1).

contains a lot more words. By looking at the default-lexicon file (Guarraci, 2011), it's possible to see exactly how SimpleNLG stores lexicon entries.

Figure 6.2 shows the entry for the adjective 'short'. Apart from the root, SimpleNLG also stores some information about the type of adjective. According to this entry, short is a classifying, predicative and qualitative adjective.

Figure 6.3 shows the entry for 'shoe', a normal noun. As English does not have a grammatical gender, plurals are easy to make, SimpleNLG only stores the root, part of speech tag and ID.

Figure 6.4 shows the entry for 'to go', an irregular verb. The particle and auxiliary verb are not stored. This is because particles are not present in English, and English always uses 'have' as an auxiliary verb for the perfect forms.

The conjugations are only stored if they are irregular. For example, the present first person singular form is not included, as it is conjugated using the standard rules, i.e. by just taking the root.

To store what type of verb to go is, the lexicon includes the tags transitive and intransitive for this verb. This means that the verb can be both transitive and intransitive. Another possible tag is ditransitive, which is not present in this verb.

The lexicon for SimpleNLG does not include the preposition or preposition target. Instead, it expects the client module, usually a sentence planner, to specify the prepositions in the input structure.

```
<word>
  <base>short</base>
  <category>adjective</category>
  <id>E0055691</id>
  <classifying/>
  <predicative/>
  <qualitative/>
</word>
```

Figure 6.2: The entry for the adjective *short* in the SimpleNLG lexicon.

```
<word>
  <base>shoe</base>
  <category>noun</category>
  <id>E0055677</id>
</word>
```

Figure 6.3: The entry for the noun *shoe* in the SimpleNLG lexicon.

Alpino

Grégoire (2007) describes the lexicon used in Alpino, a system that is used to parse Dutch text to a grammatical structure called a dependency tree. As shown by De Kok and van Noord (2010), it can also be used the other way around: to create surface text from a given grammatical dependency tree.

Alpino has several sublexicons instead of one. For example, verbs are stored in a file called *verbs.pl* and nouns in a file called *noun.pl*. Verbs in Alpino are stored using so called verb tags, which store the following properties:

```
<word>
  <base>go</base>
  <category>verb</category>
  <id>E0030073</id>
  <present3s>goes</present3s>
  <intransitive/>
  <past>went</past>
  <pastParticiple>gone</pastParticiple>
  <presentParticiple>going</presentParticiple>
  <transitive/>
</word>
```

Figure 6.4: The entry for the verb *to go* in the SimpleNLG lexicon.

- Tense and conjugation information
- Auxiliary verb
- Complements

Complements refer to structures that can be made with this verb. This can refer to constructions with a preposition, but also to fixed idioms. Figure 6.5 contains an example verb tag for the verb *draaien* (to turn), with a set of structures that contain the verb stored as complements.

Alpino stores nouns with the following properties:

- Plural
- Article (de or het)
- Type of noun (mass or count)

Figure 6.6 contains an example entry for the noun *breuk* (breach). It's interesting that Alpino stores the article instead of the gender. As discussed before, there are no differences between words that are grammatically male or female in the Dutch that is spoken in the Netherlands. So it's possible to just store the article, as Alpino does.

```
v(draai,draait,draaien,gedraaid,draaide,draaiden,
      [z([part_intransitive(om),
    part_ld_pp(om)]),
      h([transitive,
    intransitive,
1: part_fixed(om,[{[acc(hand),pc(voor)]}],no_passive),
2: part_fixed(aan,[acc(duimschroef),dat],norm_passive),
3: part_fixed(aan,[acc(duimschroef)],norm_passive),
4: fixed([acc(loer),dat],imp_passive),
    iemand een rad voor ogen draaien -->
5: fixed([[voor,ogen],[een,rad],dat],imp_passive),
    iemand een rad voor de ogen draaien -->
6: fixed([[voor,de,ogen],[een,rad],dat],imp_passive),
      b([intransitive,
7: fixed([[quite]],imp_passive),
8: fixed([[quitte]],imp_passive)])]).
```

Figure 6.5: The entry for the verb *draaien* (to turn) in the Alpino lexicon (Grégoire, 2007, Figure 1).

```
(2)  n([pl(breuken),sg(breuk)],de,[],
       [been,
        bot,
        enkel,
        contract]).
```

Figure 6.6: The entry for the noun *breuk* (breach) in the Alpino lexicon (Grégoire, 2007, Figure 2).

Cornetto

Cornetto (Horák et al., 2008) is not a language generation system itself, but a lexical database that describes the semantic relations between words. It is made by combining the data from the Dutch

Wordnet with the Referentiebestand Nederlands (RBN), a Dutch database with detailed information about words.

Cornetto consists of three parts. The first is a set of lexical units from the RBN, which contain information about word senses. These contain the linguistic information for each word. The second is a collection of Synsets, which are derived from the Dutch Wordnet. The synsets contain the semantic relations between words. The third part is an ontology defining the semantic relations between words.

For the lexicon, the lexical units are initially the most interesting. However, even though the semantic relations and ontology are not directly used for inflecting words, they can be very useful to the referring expressions generator, which could for example use a more general term for a concept to make the text more varied.

To see what kind of information is stored in the lexical units, I looked at the RBN. The RBN stores a lot more data than for example the lexicon of SimpleNLG. The RBN stores words in the form of a table, where each row is a word and each column is a property. Because the RBN stores so many properties, I will not discuss all of them.

The basic properties of each word are the identification form, which is the infinitive in the RBN, and the part of speech tag. The RBN contains eight tables: one for adjectives, one for function words and adverbs, one for verbs, one for nouns, and then four tables with examples for each category.

The RBN also includes detailed information about the inflection of irregular verbs. To do this, it includes a column that stores whether the verb is 'weak', i.e. regular, or 'strong', i.e. irregular in the past tense. It can also signify 'mixed', for words that are partially irregular or that have both a regular and irregular past tense, and 'other' for words like 'zijn' (to be) which have different forms entirely. For strong or 'other' words it stores the participle and singular simple past forms. For verbs it also stores whether they are transitive, reflexive and/or separable.

Words that have the same spelling but a different meaning are included once for each meaning. The RBN includes a meaning-id column to differentiate between these different meanings of the same word. For each word it also includes variations in form and spelling, if these exist. The RBN also includes a list of example sentences containing each word.

For nouns the RBN stores the plural, a second plural if it exists, the gender, the article and the number, as some nouns in the RBN are already plural. For adjectives it stores the comparative, superlative and the word, usually a verb, on which it is based. Interestingly, it does not store the inflected form of adjectives. Instead, the RBN stores whether an adjective is declinable or not. The word 'sexy', for which the inflected form is the same as the root, is stored as not declinable.

6.2.4 Proposed lexicon design

In this section I will use the gathered information about the Dutch language, and the solutions from the related work to propose a lexicon format for the new Narrator.

The new lexicon will be based on the lexicon of the old Narrator, to make sure old modules keep working without much required change in the code. However, some things will be different. There should no longer be a need for a *nouns.txt* and *verbs.txt*. These files only make adding adding terms to the lexicon more complicated, as there are three files that contain lexicon information instead of one. In the new format, inflection information will be stored in the lexicon file itself. To reduce the size of the lexicon, I will follow the format of SimpleNLG: only irregular forms have to be specified. If a certain form is required but not specified, the Narrator will create it using regular inflection rules. This applies to the plural form of nouns, the inflected (root + e) form of adjectives, and verb conjugations. Something similar happens with auxiliary verbs: the auxiliary verb can be specified in the lexicon entry of a verb to be either *be* or *have*. If it's not specified, the Narrator will assume *have* is to be used.

The new lexicon will be in XML format, to make the file easily readable by a Java program, and also editable and readable by humans. The lexicon consists of a number of senses. Each sense contains an ID linking it to a concept in the ontology used by the AGENT program. Each sense can contain several entries, that contain a specific word with its grammatical information. The Narrator will try to introduce more variety to the text by choosing different words each time a certain concept is mentioned.

For prepositions I will follow the format of the old Narrator, where the preposition always applies to the target node and the lexicon specifies which part of the sentence this target node is, for example location or secondary object. To make the system a bit more flexible, the new Narrator also allows a preposition to apply to the primary object. This is the case if the specified dependency label is *obj1* and a Patiens is specified in the Fabula.

As mentioned before, there is no difference between how masculine and feminine words are treated grammatically in Dutch. Therefore, the new Narrator's lexicon will follow Alpino and only specify the definite article.

Specifying whether a verb is transitive or intransitive is not needed, as the Narrator can find this information in the Fabula. If a Fabula node contains a Patiens, the verb is transitive, if it also contains a secondary object, it's ditransitive. Otherwise it's intransitive.

6.3 Characters and entities in the story world

6.3.1 Scenario

The basic unit for storing characters and entities is the scenario. The scenario contains the information of the story world: the characters, locations, items and actions. The scenario is shared between the Narrator and the serious game: the Narrator receives the scenario along with the Fabula to create a story. This scenario serves as the setting of the story, and contains the initial state of the story world.

6.3.2 Characters

For each character a mock-up description was made, including a picture and a small description in natural language. The goal of these descriptions is to provide a shared background for the characters, that can be used by everyone working on the project. In this section I will show the description for Barry, one of the loitering youths. I will then show the XML representation of Barry that is used as input to the Narrator as part of the scenario file.

Figure 6.7 shows Barry, a loitering youth. Barry is a hot-headed 15 year old. He considers himself the leader of the group. Barry gets angry easily, a trait he shares with his father. This makes it difficult for him to be at home, creating a strong need for a territory on the streets.

Though he is angered quickly, it is possible to win his trust. He can be more reasonable when talking to him alone. When things get out of hand, he will sometimes draw his knife.

Figure 6.7: The picture used to represent Barry, by *T-Xchange*.

Personality aggressive, confronting (Competitive)

Desires: his own territory, being heard

Intentions: Take charge, solve disputes, gain trust (Cooperating)

In Figure 6.8 I have included the XML representation for this character. For this character we store his name, gender and initial location, in this case the alley. We also store his role, loitering youth, and his initial emotional state. It is also possible to store his stance towards other characters, this could be used for example to store a trusting stance towards a police officer that had a positive interaction with him before. In the narration tag specific information for the Narrator is stored. In this case that is an adjective, *tough*, that can be used to describe Barry. Note that this adjective is only used by the Narrator to spice up the text. In the future, it would be better to use an adjective based on an actual property from the serious game, for example something based on the personality of the A.I. that controls Barry.

For each character, a number of actions is also stored. As this is a lot of information which is not used by the Narrator, I chose to not include it in this report.

```
1  <character id="bystander1" control="AI" perception="relion">
     <ic>
3      <name>Barry</name>
       <gender>male</gender>
5      <at><location id="alley"></location></at>
       <role>loiteringyouth</role>
7      <personality>aggressive</personality>
       <emotion>angry</emotion>
9      <stances></stances>
       <goal>loiter</goal>
11   </ic>
     <ooc></ooc>
13   <presentation>
       <narration>
15       <adjective>tough</adjective>
   % <descriptive lang="NL">een schoffie</descriptive>
17     </narration>
     </presentation>
19 </character>
```

Figure 6.8: The XML representation of the character Barry, the loitering adolescent.

6.3.3 Locations and items

Apart from characters, the scenario file also stores locations and items. Figure 6.9 shows the information for the main street. It stores the name of the location as used by the serious game, and the paths from this location to other locations. The sense tag is really important for the Narrator: this stores the lexicon sense that should be used to refer to this location.

Figure 6.10 shows an example item, a shoe. This shoe has an initial location: the main street. As with the location, the sense tag is used by the Narrator to look up the word that should be used in the lexicon.

```
<location id="main_street">
  <name>Hoofdstraat</name>
  <path_to>
    <locations>
      <location id="alley"></location>
      <location id="square"></location>
    </locations>
  </path_to>
  <presentation>
    <narration>
      <sense>main_street</sense>
      %<descriptive lang="NL">de hoofdstraat</descriptive>
    </narration>
  </presentation>
</location>
```

Figure 6.9: The XML representation of the main street.

```
<item id="schoen1">
  <at><location id="main_street" /></at>
  <presentation>
    <narration>
      <sense>shoe</sense>
     % <descriptive lang="NL">de schoen</descriptive>
    </narration>
  </presentation>
</item>
```

Figure 6.10: The XML representation of a shoe.

Chapter 7

System overview of the new Narrator

7.1 Introduction

In this chapter I will provide an overview of the implementation of the new Narrator. I will discuss the modules that are used in the Narrator. While the report so far has focused on the Narrator from a design perspective, in this chapter I will also include technical and implementation details about each module.

7.2 Architecture

The new Narrator uses the three-stage model defined by Reiter and Dale (2000). As mentioned in Section 2.3, this is the most practical approach for when the content of the text is already known. This is also the same structure as the existing Narrator, which makes it easier to reuse existing code.

The new Narrator is based heavily on the existing Narrator made by Slabbers (2006). The Document Planner, Sentence Planner and Surface Realizer were reused from the existing system. I added Focalization, Flashback and Introduction modules to the Document Planner, and changed the graphical user interface. As the data formats used for input were changed, I created new modules for data input. The existing Document Planner, Sentence Planner and Surface realizer were all modified to accommodate for the changes in data format and added functionality, as well as fixing some smaller problems. Like the existing Narrator, the new Narrator is written in Java.

To make interaction with the Narrator easy, there will be one 'Main' class that handles all interaction. Interaction with the Narrator can happen in one of three ways:

1. Call the Narrator as a method from a Java program.

2. Run the Narrator from a command-line interface, so that it writes text to the standard output.

3. Run a graphical interface of the Narrator. This is especially useful for testing.

The Main class thus receives the Fabula data, which it then passes on to the Document Planner. The Document Planner uses it to make a general story plan containing plot elements and their relations. This plan is passed over to the Sentence Planner in the form of a Rhetorical Dependency Graph. The Sentence Planner then transforms each plot element in the graph into a Dependency Tree containing information about the structure of each sentence. This new Rhetorical Dependency Graph containing Dependency Trees is then passed on to the Surface Realizer, which performs referring expression generation and inflects words to create the final surface text.

A diagram containing the structure of the Narrator, including what data structures are passed on by the different modules, can be found in Figure 7.1.

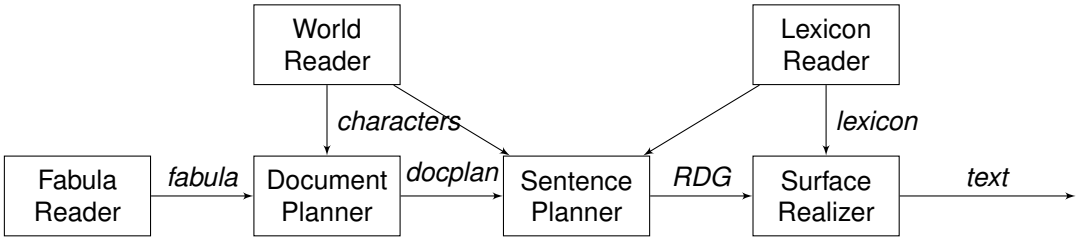

Figure 7.1: Diagram of the general structure of the Narator.

7.3 File I/O

To account for possible changes in the file format, all file I/O is separated from the rest of the code and implemented in Reader classes. If the file format changes, only the Reader class should be adapted.

FabulaReader The FabulaReader is called at the very start of language generation by the Initial Document Plan Builder, and converts a GraphML Fabula to a directed graph. It then reads the properties of each Fabula node and edge, and adds them to the graph. As a Fabula node represents an event in the story, it contains a function to easily convert it to the Plot Elements that are used by the Document Planner.

Some Fabula nodes, in particular goals and perceptions, can have sub-Plot Elements. They are used for Goal elements to store the event that the character to which a goal belongs wants to happen, e.g. when the police officer wants to talk to the youth, the sub element contains the event 'the police officer talks to the youth'. For Perceptions, they are used to store the story element that is perceived. These sub-Plot Elements are present in the Fabula as separate nodes, and connected by a special edge. In the final graph however, they should not be present as nodes but instead as attributes of their parent element. To achieve this, the Fabula Reader checks each edge in the graph. If it's a 'sub-Plot Element' edge, it creates a Plot Element out of the sub node, and adds it as an attribute to the parent element. The sub node and the sub-Plot Element edge are then deleted from the graph.

WorldReader The World Reader is an XML parser that is based on Java's SAX Parser (Oracle, 2013). It reads the scenario file provided by AGENT, and reads the characters, items and locations. These entities are then stored in a CharacterInfo object (see section 7.5) as CharacterModel objects.

LexiconReader Like the World Reader, the Lexicon Reader is also a SAX Parser. It reads the XML lexicon as described in 6.2 and stores each word as an Entry object inside a Lexicon object, which maps abstract senses to one or more word entries.

7.4 Document Planner

The document planner is responsible for transforming the Fabula that describes the story events into a Document Plan, a tree structure that contains the events as plot elements and the relation between them. This Document Plan is then passed on to the Sentence Planner. The Document Planner uses several submodules to achieve this, which will be each discussed separately in this section. The structure of the Document Planner, including its submodules, can be found in Figure 7.2.

Figure 7.2: Diagram of the structure of the Document Planner.

Focalization module This module filters the initial graph to only contain nodes that contain the character from whose point of view the story is told. The algorithm is discussed in Section 4.3. The focalization module is new in this version of the Narrator.

Flashback module If the flashback setting is enabled, this module searches the history for flashbacks to add to the story. The algorithm is discussed in Section 5.3. The flashback module is new in this version of the Narrator.

Initial Document Plan Builder The Initial Document Plan Builder (IDPB) transforms the Fabula into an initial document plan. As the Fabula has a graph structure instead of a tree structure, this is not straight forward. The IDPB does this by reading the Fabula file as a graph. It then transforms this graph into a tree using an algorithm made by Slabbers (2006, Paragraph 6.2.2). Meanwhile, it creates Plot Elements out of Fabula nodes, and Rhetorical Relations out of Fabula edges. Though the basic algorithm of the IDPB was not changed, it was heavily modified to support the new data format.

Introduction Module The introduction module writes a little introduction for each character. Currently, it just adds a Plot Element to the start of the story that contains for each character their location. The Background Information Supplier then adds the names of each character to their introduction. If focalization is enabled, it only introduces characters in the same location as the focalized character.

The introduction module is new in this version of the Narrator. An example of an introduction can be found in Figure 7.3.

```
Een mooie omstander, die Manuela heet, is op een plein.
Een lange hangjongere, die Richard heet, is in een hoofdstraat.
Ik ben ook in de hoofdstraat.
Een politieman, die Adrie heet, is in een steegje.
```

Figure 7.3: An example introduction, focalized from the point of view of one of the loitering youths.

Background Information Supplier The Background Information Supplier (BGIS) is responsible for adding background information about characters to the document plan. Currently it tells the name of a certain character, as in the example sentence "the police officer, *whose name is Adrie*, talks to a loitering boy". Mentioning the name is important for the story, as the referring expressions generator will sometimes choose the name of a character to refer to them. Without introducing this name first, the reader will not know to who it's referring to.

To avoid telling a fact a few times in a row, the BGIS keeps a discourse history of facts that have been told. The discourse history will prevent the BGIS from telling a fact twice most of the time, however it will repeat facts to refresh the reader's memory if the story is long.

The new BGIS does not differ significantly from the one created by Slabbers, the only difference being that it does not mention the name of the character from whose perspective the story is told, if focalization is enabled and the perspective is either first or second person. It's worth mentioning that Slabbers' BGIS was able to tell more types of facts than just the name, such as a fact about a character or a general fact about the story world. These facts were however hard-coded for the fairy-tale domain, and not re-added for the loitering youth domain.

7.5 Sentence Planner

Template transformer The microplanner works by transforming each plot element from the document plan into a sentence plan by using templates that are specified in the code. It also fills in the lexicalization and character information by using a Lexicon and CharacterInfo class, as mentioned below. The templates used in the sentence planner were not changed and are therefore the same as in the old Narrator (Slabbers, 2006, Section 7.2). The templates can be found in Figure 7.4.

Lexicon The Lexicon links concepts to words and linguistic information, such as the determiner for nouns. It uses a module called Lexicon Reader, which reads the XML file that contains the lexicon and transforms it into a set of entries for easy use within the code. More information about the lexicon format can be found in Section 6.2. The Lexicon and Lexicon Reader modules were recreated to support the new data format.

LexicalChooser The LexicalChooser is called when the Microplanner needs to choose a word for a certain concept. Since some concepts can be expressed using multiple words, this choice is more difficult than just calling the lexicon. The LexicalChooser contains a word history to remember which words were already chosen, and tries to create variety in the story by picking words that were not used yet with a higher probability. It always selects the first option the first time a concept is mentioned. For the next times the concept is mentioned it increases the probability for the other options.

CharacterInfo This class reads information about characters, and also objects in the story world. The Microplanner can then query this class to find out properties of characters. This stores for each character: The systematic name used in the Fabula, a class this character belongs to (such as bystander or loitering youth), this character's gender (or Neutral) and their name. More information on the data format can be found in Section 6.3.

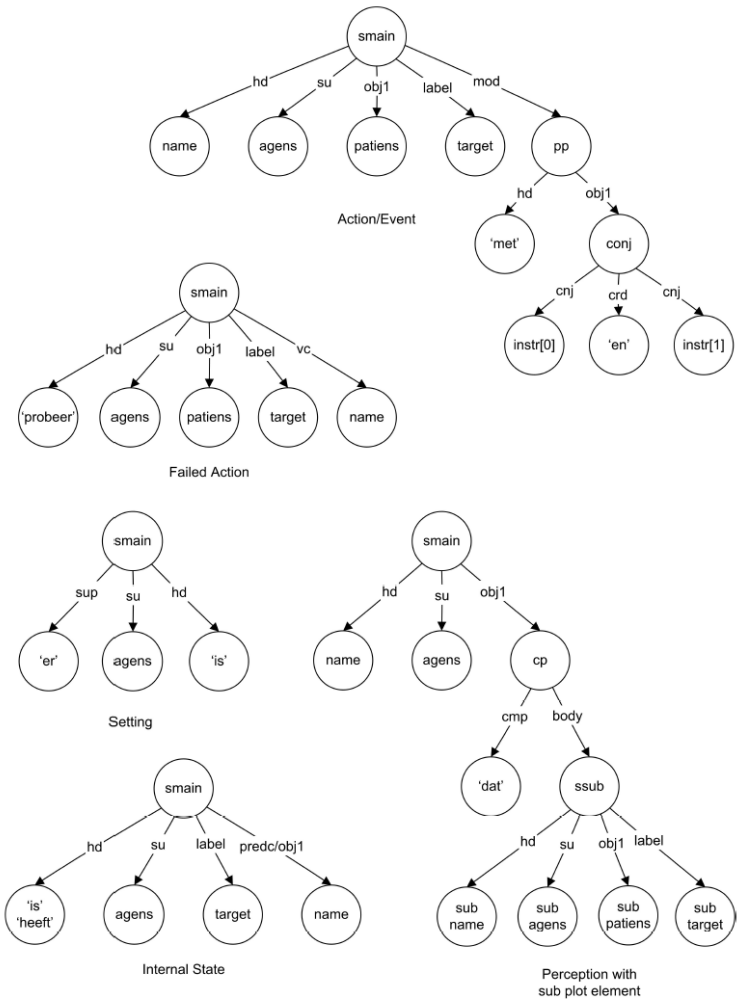

Figure 7.4: The templates that are used in the Sentence Planner (Slabbers, 2006, Figure 7.2).

7.6 Surface Realizer

The surface realizer is responsible for converting the dependency trees created by the Microplanner into natural language. It is based on the surface realizer created by Hielkema (2005), with changes by Nanda Slabbers and myself. It consists of two parts: the referring expressions generator, and the surface realizer itself.

7.6.1 Referring Expressions Generator

The REG transforms an entity in an expression referring to that entity. To do this, it has to choose between a pronoun, the name of the entity, or the role of a character (‘the policeman’). If the entity that has to be referred to is the focalizing character, and the Narrator is focalizing in first or second person, the REG always chooses ‘I’ or ‘you’ respectively. For the other cases, it uses a modification of Dale’s Incremental Algorithm made by Krahmer and Theune (2002).

It works together with the FabulaChecker, InferenceModule and Hierarchy to use ontology information when it is available. It uses the LexicalChooser for converting concepts such as roles into words. For example, for a character that has the role ‘bystander’, the word bystander is looked up in the lexicon, which can result in the referring expression ‘een omstander’ (a bystander). To avoid using the same expression in a row, the REG keeps a history of what expressions have been used in the story so far.

FabulaChecker This class looks at the story world to find information about characters and items that could be useful for the referring expressions generator. For example, if an object is the only object of a certain type, or a character the only of a certain role, it should be referred to with a definite article. It also looks at the relations between objects and characters, for example by specifying that a certain object belongs to a certain character.

As no ontology is currently specified for the new Virtual Storyteller, this module currently does not do anything.

InferenceModule This class is used to apply inference rules when generating referring expressions. Inference rules are rules that transform multiple premises into a conclusion. To do this, it uses the FabulaChecker as a knowledge base. The rules are read from a file called *rules.txt*. Currently this file does not contain any rules that apply to the new domain, so this is not used currently.

PossibleAdjectives This module links possible adjectives to nouns, and possible adverbs to adjectives. To do this, it reads which adjectives can be used to describe a certain noun. It's closely related to the class ChosenAdjectives, which stores what adjectives have already been used for a certain concept, so that the Narrator can add a bit of variation of word choice if possible.

Adjectives are read from the story world file. Each character and item contains a set of properties that apply to it, such as *timid* for a loitering youth or *heavy* for a bag that one of the characters is carrying. The properties are stored as word senses that link to information in the lexicon. The adjectives are added to the text by the Referring Expressions Generator: when it's creating a noun phrase for a noun that has a matching adjective, there's a 25% chance it will add that adjective to the phrase.

Possible adverbs are read from the lexicon. Each adjective specifies which adverbs are possible to add, for example *heel* (very) for the adjective *mooi* (beautiful).

Hierarchy This models a hierarchy of entities based on different relations: part of (corner is part of street), belongs to (iPhone belongs to a loitering youth), and relation (Henk is the uncle of Barry). This currently does not work, as no ontology is yet specified for the new Virtual Storyteller.

7.6.2 Surface Realizer

This part is responsible for creating the actual surface text. It uses a set of submodules, which are discussed here.

Conjunctor This class creates a grammatical Dependency Tree out of a relation and a key word.

Relation Transformer This transforms a dependency graph into one or a set of dependency trees, which can be connected by a cue word.

Orthography Adds punctuation and capitalization to the final text to make proper sentences. Currently only capitalizes the first word and adds a period to the end of the sentence.

Morphology This inflects the root word of a dependency tree node. It can inflect verbs, nouns, determiners and adjectives. For other words, it just copies the root word. It can inflect irregular words by reading data from the lexicon, and regular words by using a set of regular inflection rules. To inflect words, it uses the libraries mentioned below.

Eliptor This combines nodes that are similar using an ellipsis construction. This refers to combining two sentences, and removing some information so that identical information is only mentioned once. An example of ellipsis is the combination of the sentences *Joe goes to Amsterdam* and *Mary goes to Amsterdam* into the sentence *Joe and Mary go to Amsterdam*.

7.6.3 Morphology libraries

The surface realizer uses a number of libraries to correctly inflect different types of words, and to decide on the order of sentence parts.

LibraryConstants This interface contains all the constants that have to do with the dependency tree and the inflection of words. It lists for example part of speech tags, dependency relations and syntactic categories.

AdditiveLib, ContrastLib, PurposeLib, RelativeLib, TemporalLib These libraries deal with adding the right cue words to their respective rhetorical relations.

SMainLib This library deals with the ordering of sentence parts. To do this, it has a rule for each combination of parts. For example, the combination of a subject, and object and a verb is ordered as "subject, verb, object". As there are a lot of different combinations, this library has a lot of rules: over 100.

VerbInflecter This module inflects verbs. If possible, it uses a given form from the lexicon. If this form is not found, it uses an algorithm with grammatical rules to inflect the verb. The current Narrator uses the algorithm by Hielkema (2005, Figure B.2), with the following exceptions: if the lexicon contains a singular simple past form, but no plural, the plural form is created from the singular form by adding 'en'. For example, if 'zag' is specified as the singular past of 'zien' (to see), the VerbInflecter creates 'zagen' as the plural past form.

The second exception is the infinitive. Hielkema (2005)'s algorithm creates this from the root, but the VerbInflecter will use the plural present form instead if it is specified. The Verb Inflecter was changed from the original version to support reading inflected forms from the lexicon.

NounInflecter, AdjInflecter The NounInflecter and AdjInflecter inflect nouns and adjectives, respectively. They use the same approach as the VerbInflecter: if the plural form (for nouns) or the inflected form (root+e, for adjectives) is found in the lexicon, they will use that. Otherwise, they use Hielkema's algorithm. The algorithm for nouns is specified in Hielkema (2005, Figure B.1). The adjective algorithm was not specified in Hielkema's report. Both the Noun and Adjective Inflecters were changed from the original version to support reading inflected forms from the lexicon.

7.7 Graphical User Interface

To make testing for me and future users easier, I have created a GUI that lets the user choose a Fabula file as input and set various settings that have to do with the language generation process.

The GUI was written in the Eclipse Standard Widget Toolkit (Northover and Wilson, 2004). This toolkit allowed me to easily create a GUI that looks native on Windows, Mac and Linux. Compared to Java's Swing toolkit, it was easier for me to use native widgets such as the native file dialog. This results in an interface that feels more integrated with the rest of the operating system to the user, as it looks and feels the same as most other applications.

The input window is divided into two tabs: one with the file input box and basic settings, and one with advanced settings. This keeps the advanced settings initially hidden, cleaning up the interface, but allows for easy access when needed. Figure 7.5 contains a screen shot of the Narrator's GUI in Windows 8.1.

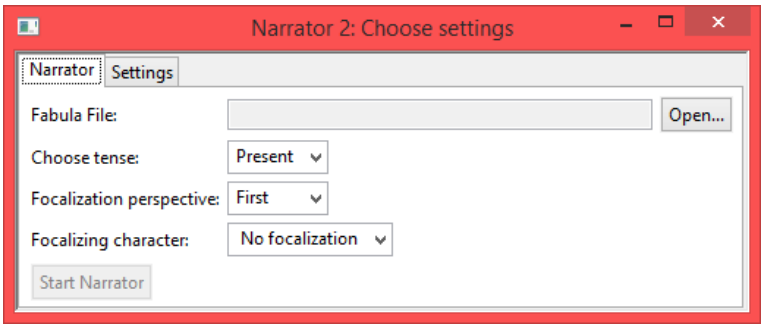

Figure 7.5: The Narrator's GUI on Windows 8.1.

7.8 Connection with AGENT

To connect with the serious game, a separate system was made. This system can connect with a server running the serious game, receive the events from the game, and then call the Narrator to create a text in natural language from these events.

AGENT's serious game was already designed with communication with other programs in mind. It does this by using a STOMP-server to relay messages about games, the system itself, and the actions performed in each game. The reason behind this was that multiple types of interfaces are able to connect to the game. The game has its own simple 2D interface, which shows what happens in the game, but it can also connect to a Re-lion interface, which allows the player to be fully immersed in a 3D environment.[1] In order for the Narrator to create text from the events in the story, it should connect in the same way as a user interface.

The connector works as follows. The user can connect to a STOMP server using a GUI (shown in Figure 7.6) that is similar to that of the off-line Narrator. Once connected, the user can select a game and tell the Narrator to start listening to that game. The server then sends the scenario file, which is

[1] See http://www.re-lion.com/blacksuit.html

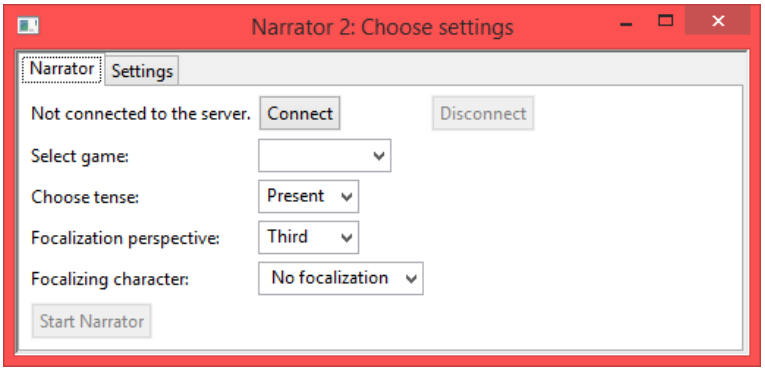

Figure 7.6: The GUI for the AGENT connector on Windows 8.1.

saved by the Narrator. While the simulation runs, the server sends actions performed by characters and players in the form of `action_response` messages. If present, causes for events are also sent in the form of `action_cause` messages. The user can then request that the Narrator creates a story. This is done by converting the saved messages to Fabula nodes and edges, creating a Fabula file, and then using that as input.

Currently the connector is not completely implemented, as not all functionality is present in the serious game server yet. It can currently receive actions from the serious game and create a story with these actions. It is however not yet able to receive other story elements, such as goals and emotional states. It is also not able to receive the Fabula edges that store the causal links between story elements.

Chapter 8

Evaluation

In this chapter I will discuss the evaluation of the Narrator. I start by surveying related work as inspiration for my own evaluation. Afterwards I present the evaluation of the Narrator, split in two parts: an analysis of the texts generated by the Narrator, and evaluation through a questionnaire, filled in by human judges. For both I present the setup of the evaluation, the results, and a discussion of the results.

8.1 Related work

8.1.1 Dale & Mellish

Dale and Mellish (1998) describe the problems of evaluating Natural Language Generation (NLG) Systems, compared to systems that analyze natural language. They formulate a set of questions that a researcher should ask themselves when evaluating an NLG system. Dale & Mellish recommend looking at the input to the system, i..e the data that is used to generate text. In the case of the Narrator, the input is a combination of the Fabula, the world file and the lexicon. The input should be complete, but should not contain information that 'nudges' the NLG system to a better solution. An example of this is hard-coding information for one specific text, making that text artificially better.

It is also important to realize what the output should be, as there are no objective measures for measuring how good a text is, and often there are multiple right answers. Finding out what to measure can therefore be really difficult. A metric such as similarity to a human-written text can be useful, but human judges don't always agree on what the reference text should be. Some measures have to be put into context. Dale and Mellish give an example of a system that fills in the article for an English text with missing articles. A baseline system that always guesses 'the' got a score of 67% correct, while human judges could not get higher than 95%. Without these upper and lower bounds as context, the measure of how much the system got right would not be very useful.

When evaluating output text using human judges, several problems arise. Different people can have different definitions of subjective measures such as 'coherence' and 'readability'. In other cases, judges disagree on what the 'right answer' should be. When using human judges, it can be useful to investigate what causes disagreement, such as one judge giving strange answers, or having one question on which judges disagree a lot. Dale and Mellish also recommend using the kappa statistic as a measure of agreement between judges.

8.1.2 Reiter & Belz

Reiter and Belz (2009) present a research in which they compare automatic metrics to evaluations based on human judges. Reiter & Belz define three different types of evaluation methods for NLG systems. *Task-based evaluation*, in which users have to perform some task based on generated text, e.g. following instructions. The score is then how good they are at performing that task. *Evaluation based on human ratings*, in which human judges rate a text on criteria such as fluency, coherence and style. And finally *Automatic evaluations*, based on metrics which compare the generated text to one ore more human-written reference texts. In this research, the second and third categories are compared.

To do this, Reiter & Belz used a system that can automatically generate weather reports for offshore oil rigs. They created multiple algorithms for generating these reports. The texts generated by these different algorithms were then evaluated by expert users, i.e. people who had experience reading these reports, and non-expert users on both linguistic quality and content quality. The texts were also evaluated by automatic algorithms that determined the similarity to a set of reference texts.

They found that the NIST-5 and BLEU-4 metrics had a significant correlation with linguistic quality, but not with content quality. They conclude that using automatic metrics is useful for evaluating linguistic expression, but not for evaluating content determination.

8.1.3 Harbers, van den Bosch & Meyer

Harbers et al. (2010) describe a different kind of research, from the field of explainable A.I. They made a scenario-based training system using fire-fighting training as a domain, using BDI (Belief, Desire, Intention) agents as fellow firefighters in a team. The agents' behavior is specified using hierarchical goals, this means that the agent will try to accomplish goals, and that goals can have one or multiple subgoals. Different goal hierarchies exist for the different roles agents can take, for example a leading role.

After running the training system, it generates a behavior log in the form of a list of goals that were achieved and actions that were performed. Goals are linked to their respective parent goals. Harbers et al. then made four different algorithms for describing the behavior of the agent. Unlike the other systems I have reviewed, their system does not generate natural language. Instead, Harbers et al. use four different algorithms that only choose the goal or action that caused a certain action. The first (A) tells the parent goal that caused an action. The second (B) tells the goal two levels up the goal hierarchy. The third (C) tells the belief that enabled the execution of the action. The fourth algorithm (D) works by telling the first action or goal that follows the action.

The explanation algorithms were evaluated by having each algorithm generate an explanation for a given action. Human judges then chose which explanation they found was the most useful, which was explained as 'which explanation helps you best in understanding the leading fire fighter's motives for performing these actions?'. The actions that had to be explained were divided into four categories, based on the relation of subgoals to the parent goal:

All All subgoals have to be completed before the parent goal, in no particular order.

If Some subgoals have to be completed, based on some condition that depends on the environment.

One Any one of the subgoals can be completed.

Seq All subgoals have to be complete in a specific order.

Their evaluation found that the best explanation depends on the type of relation. For *if* and *one* type goals the judges greatly preferred Algorithm C (enabling belief), whereas for types *all* and *seq* Algorithm A (parent goal) and B (grandparent goal) were also chosen relatively often. Algorithm D (following goal or action) was rarely chosen.

8.1.4 StoryEval

Rowe et al. (2009) describe StoryEval, a framework for evaluating automatically generated stories. They describe four different ways of evaluating a story.

The first is evaluation by narrative metrics, this refers to evaluating properties of the text to assess the narrative quality. Rowe et al. give several examples of how to evaluate using narrative metrics. One option is performing a human participation experiment, i.e. showing human judges the generated text and asking them to rate them on points such as style, readability, grammar and diction. Another option is using an objective function to automatically measure the properties of the narrative.

The second method is a cognitive-affective study, which refers to measuring the cognitive reaction of the reader to a story. Specifically, this can be done by evaluating the presence, i.e. sense of being part of the story, engagement and emotional experience. This could be done by asking users to periodically report on their emotions during the narrative experience, or to use a questionnaire afterwards. Rowe et al. mention however that both these techniques are highly subjective, as it is difficult to directly measure an emotional experience.

The third method, called director centric evaluation has a director agent evaluate the story while it is being told. This type of evaluation would have to be part of the Virtual Storyteller instead of the Narrator, as that contains the multi-agent system generating the events of the story. A director agent would have to be an agent in that system. The last method, extrinsic narrative evaluation, is about evaluating the (educational) goals of the narrative. This is very important for systems that are used for education and training.

8.1.5 McIntyre & Lapata

McIntyre and Lapata (2009) describe a method of creating stories by combining concepts from a database, without needing a pre-generated plot beforehand. The automatic evaluation of stories is an important part of their system. It uses a generate-and-test method which first creates a lot of different stories, and then ranks them on how interesting and coherent they are.

To measure the 'interestingness' of a story, the authors have created an interest model by analyzing a corpus of stories (Aesop's fables), and finding a relation between lexical features and interestingness. This is done by using an SVM model. Their method for measuring coherence is based on 'local coherence', which is the coherence from sentence to sentence. This local coherence is measured using the

Entity Grid approach (Barzilay and Lapata, 2005), trained on fairy tales from the Andrew Lang collection (Lang, 1889-1910). Randomly shuffled versions of the same stories were used as examples of incoherent stories. Their method could differentiate between the original and shuffled stories with a success rate of 67%.

Afterwards, the system was evaluated by human judges. 30 random stories were generated by the system, which were rated on a scale of 1 to 5 on *Fluency*, *Coherence*, and *Interestingness*. The result was that the rank-based system created significantly more interesting and coherent stories than just random stories.

8.1.6 Callaway & Lester

Callaway and Lester (2002) describe STORYBOOK, a system for automatically generating narrative prose. Their system was used to create fairy tales. In Section 2.2 I describe the internal structure of STORYBOOK in more detail.

By disabling or enabling specific components, the system was used to create a set of different versions of Little Red Riding Hood. The components that were enabled or disabled were the discourse history, which is used to produce pronouns, the revision system, which is used for clause aggregation, and the lexical choice module, which introduces variation in word choice.

The generated stories were evaluated by human judges on the following nine points:

Overall How good was the story compared to other fairy tales?

Style Was the style appropriate for fairy tales?

Grammar How good was the syntactic quality of the story?

Flow How well did sentences flow over into the next sentence?

Diction How interesting or appropriate were the word choices?

Readability Was it difficult or easy to read the story?

Logicality Did the story include all crucial information and was it in the right order?

Detail Did the story have the right amount of detail?

Believability Did the characters behave in a believable matter?

8.1.7 Igartua

Igartua (2010) describes something completely different. In his paper he researched the importance of identification with characters in media. To measure the identification with characters, Igartua used the EDI scale (Igartua and Páez, 1998), a survey of 14 questions which are answered on a scale of 1 (not at all agree) to 5 (very much agree).

Igartua presents three different studies on this topic. In the first study, a survey was held on visitors of a comedy, thriller and drama movie, measuring their mood and the identification with the characters. The researcher found that identification with characters played an important role in how much viewers enjoyed a movie.

In the second study, participants were asked to watch a dramatic movie on labor exploitation of immigrants. The movie was chosen because it was a serious movie that was supposed to make people think about the subject matter. Igartua found that identification with the characters correlated with a greater effect on the mood of the viewers. He also found that identification had a positive association with how much viewers reflected on the subject matter of the movie. Identification with the characters resulted in a greater cognitive elaboration.

In the third study, the participants watched a political *mockumentary*, on the lives of Mexican immigrants in the U.S.A. Their views on immigrants before and after the movie were measured. Igartua found that the movie had a positive effect on the viewers' attitude on immigrants. He also found that this positive effect was greater if the viewers identified with the characters more. He concludes that identification with the characters plays an important role in the persuasive impact of media.

8.1.8 Discussion

In the literature I found broadly two different types of methods of evaluating an NLG system: automatic methods, using rules to calculate some value about the quality of a text, and methods using human judges to evaluate the quality of a text.

An automatic evaluation could be used to evaluate the narrative metrics of a story, as mentioned by Rowe et al. (2009). However, it is hard to find tools that could be used to evaluate Dutch text as written by the Narrator. This same problem applies to McIntyre and Lapata (2009). Their rules for evaluating lexical features would have to be recreated for Dutch, which falls outside the scope of this project. It

might be possible to use their coherence model to evaluate Dutch stories, but the different method of generating stories in the Narrator makes stories that are already a lot more coherent than their randomly generated stories. Reiter and Belz (2009) describe automatic metrics like NIST-5 that could be useful for evaluating the linguistic quality. Unfortunately, these all rely on comparison with reference text, which does not exist for the texts the Narrator generates.

Harbers et al. (2010) evaluated their project by asking human judges what they thought the cause for a certain action was. This method is applicable to the Narrator, as the Fabula model is based on causes and effects of characters' action. It could be used to evaluate flashbacks and focalization by measuring their effect on the explainability of the A.I.'s actions.

Rowe et al. (2009) mention using a cognitive-affective study, which is also a form of human evaluation. It can be very useful for the Narrator to measure this emotional experience. Igartua (2010) found that identification with the main characters of media has a positive effect on reflection and persuasive impact. If this translates to the domain of the Narrator, more identification with the characters could have a positive impact on the educational value. Focalization can have an effect on the identification, as it makes the reader experience the story from the point of view of a certain character. It can therefore be useful to test the effect of focalization on the identification with the focalizing character.

A human evaluation can also be used to evaluate narrative properties of the story, such as style and fluency. This is done by both Callaway and Lester (2002) and McIntyre and Lapata (2009). It makes sense to evaluate the Narrator this way as well. If a human evaluation is done anyway, it's not a lot more work to include these narrative properties as well, and they could tell us a lot about the quality of the generated text.

Finally, Rowe et al. (2009) mention evaluating the educational goals of the generated text. This would be really interesting for the Narrator in relation to the serious game. However, this is not possible as long as the serious game is not finished yet.

8.2 Analysis of generated text

8.2.1 Evaluation setup

In this first part of the evaluation I test the Narrator by having it create a few stories from manually created Fabula files. I then look at what goes right and what goes wrong. I try to use input that 'covers all bases', i.e. that uses as much of the Narrator's functionality as possible. I also try to use cases where something could easily go wrong, in order to test the Narrator well.

8.2.2 Analysis of stories

Story 1

```
Een omstander, die Manuela heet, is in een hoofdstraat.
Ook is een omstander, die Richard heet, in de hoofdstraat.
Ook is een hangjongere, die Barry heet, in de hoofdstraat.
Ook is een politieman, die Adrie heet, in de hoofdstraat.
Hij wil met Manuela praten.
Barry wil dat de politieman met haar niet spreekt, dus met
een schoen probeert de omstander de politieman te slaan.
Omdat zij niet meer in de hoofdstraat wil zijn, loopt zij
naar een plein.
```

Figure 8.1: A story with three different types of goals.

In Figure 8.1 I include a sample story with three types of goals: an *attain goal*, i.e. I want something; an *avoid goal*, i.e. I don't want something; and a *leave goal*, i.e. I don't want this thing anymore. There is a fourth type of goal in the Narrator, the *sustain goal*, but this is currently realized in the same way as an attain goal so it was not necessary to include it. The story also includes a few actions caused by these goals.

The story starts by telling the location of the characters, in this case the main street. The first time this main street is mentioned, the Narrator uses the indefinite article *een* (an), it would be better to use the definite article *de* (the) right away.

A few sentences in this story have the wrong word order. De sentence *'Barry wil dat de politieman met haar niet spreekt'* should be realized as *'Barry wil dat de politieman niet met haar spreekt'*, while the next sentence should be *'dus probeert de omstander de politieman te slaan met een schoen.'*.

Story 2

```
Een mooie omstander, die Manuela heet, is in een hoofdstraat.
Ook is een lange omstander, die Richard heet, in de hoofdstraat.
Ook ben ik in de hoofdstraat.
Een politieman, die Adrie heet, is op een plein.
Ik zie dat de politieman naar de hoofdstraat gaat, en eerder
heeft hij mij beledigd.
Ik ben boos en ik schreeuw naar de politieman.
```

Figure 8.2: A story with focalization and a flashback. The story is told from the perspective of Barry, the loitering adolescent.

In Figure 8.2 I show a story with both flashbacks and focalization enabled. The flashback event is properly copied from the history file, and is in the perfect tense, as it should be. The story itself does not contain any events that do not include the loitering youth or are perceived by him. Barry, from whose perspective the story is told, is always referred to with first person pronouns, and the verbs are also inflected in first person mode. In this case, focalization and flashbacks work well even when used together.

Something strange happens here with the aggregation of sentences. While it's not grammatically wrong, it would be better if the clause *'eerder heeft hij mij beledigd'* (before, he has insulted me) was connected to the clause *'ik ben boos'* (I am angry) with a causal relation. In the Fabula, being insulted causes the loitering youth to be angry, that could be communicated better in this story.

Story 3

```
Een politieman wil met een stoere hangjongere, die Barry heet, praten.
Hij heet Adrie.
Daarom gaat hij naar een hoofdstraat.
Hij beledigt Barry, zodat hij boos is.
Een mooie omstander, die Manuela heet, gaat naar de hoofdstraat en,
doordat hij de politieman met een schoen slaat, is hij verdrietig.
Zij ziet dat de hangjongere hem met de schoen mept.
```

Figure 8.3: A simple example story with three characters.

In Figure 8.3 I include another sample story, this time without an introduction telling the location of the characters first. In this case, the Narrator adds the names of characters as elaboration relations the first time they are mentioned. This is strange for the first sentence, in which the Narrator decides that telling the name of the police officer does not fit in the sentence. Instead the name is told in the next sentence, which looks awkward and breaks the causal connection with the sentence after that.

In this story, some things go wrong with referring expression generation as well. In the fourth sentence ('Hij beledigt Barry, zodat hij boos is', 'He insults Barry, so he is angry') the first 'hij' (he) refers to the police officer, while the second refers to the loitering youth. This last occurrence is ambiguous: it could refer to the police officer too, which could leave the reader confused about who has just become angry.

In the next sentence, something similar happens. Manuela goes to the main street, while Barry hits the police officer with a shoe. In this sentence, a pronoun is used to refer to Barry, while the reader might expect any pronoun here to refer to Manuela. In this case it looks especially weird, as the reader will expect female pronouns to be used for someone called Manuela!

Finally, it would be better if the shoe Barry uses to hit the police officer belonged to Barry. It makes sense that the loitering youth hits the officer with his own, shoe, but there is currently no way for the Narrator to specify ownership.

8.2.3 Discussion

Ontology The stories generated as examples could be improved by including an ontology to teach the Narrator some more information about entities. For example, the stories now always use an indefinite article the first time they refer to a new entity. This is good when talking about a concept, for example a bystander, of which there is more than one. For entities which are the only one of a concept, such as the main street, the Narrator should however use a definite article. By simply including whether an entity is unique this improvement could be made. The ontology could also include ownership. By doing this,

sentences such as *'de omstander slaat de politieman met een schoen'* (the bystander hits the police officer with a shoe) could be improved by saying *'zijn schoen'* (his shoe).

Referring expression generation The generation of referring expressions in the Narrator can be improved. While it does not choose grammatically or semantically wrong referring expressions, it sometimes makes strange choices on whether to use a pronoun, or a name or role. A future version of the Narrator could be improved by looking at the algorithm that chooses whether a pronoun is used or not.

Word order The story in Figure 8.1 has a few obvious problems with word order, that are not easily fixed. The current Sentence Planner does not specify the order of words, it instead specifies the concepts in the sentence and their role. The Surface Realizer then uses a long list of rules to decide the order of a sentence. This goes well for simple sentences with a Subject, Verb and Object, but it goes wrong when a sentence has modifier words such as 'niet' in the sentence *'Barry wil dat de politieman met haar niet spreekt'*.

The problem is that different modifier words go in different places in the sentences. A solution, proposed by Hielkema (2005), is to create more categories of modifier words. This would however cause an exponential increase in the already large (over 100) set of order rules. Another option would be to give the Sentence Planner more control over the order of words in the sentence, resulting in a more template-like structure instead of a parse tree as input for the Surface Realizer. This would however require a redesign of the Narrator.

Document planning The Document Planner is the part that is, among other things, responsible for converting the Fabula graph into a binary tree with the relations between plot elements. Unfortunately, this part sometimes breaks up causal relationships that are important for the understanding of the story. While I do not know how this could easily be fixed, I think the quality of the generated stories can be improved by better document plans.

Known bugs and other problems The Narrator crashes when the Fabula contains only one node. This has to do with the basic design of the Narrator, and is not easy to fix as a lot of Narrator modules expect a document plan with at least one relation containing a nucleus and a satellite. With only one plot element, no such relation can be made. This problem also occurs when the resulting story after focalization only contains a single plot element. Currently this is 'fixed' by presenting the user with an error explaining that a Fabula file should have at least two nodes.

8.3 Human evaluation

8.3.1 Introduction

The second part of the evaluation consists of a human evaluation, using an on-line questionnaire in which human judges are asked to rate the generated stories. This questionnaire consists of three parts: a text quality part, an identification part, i.e. how much do the judges identify with the characters in the stories, and an understanding part in which judges are asked what they think the cause of a certain story event was.

In Section 8.3.2 I will first formulate a few hypotheses for what I think the result of the human evaluation will be. In Section 8.3.3 I will then describe the experiment in more detail. In Section 8.3.4 I will describe the results, which I will discuss in Section 8.3.5.

8.3.2 Hypotheses

For the text quality evaluation, I expect that the Narrator will do fairly well for grammar, but not as well for fluency or style, as it is hard to achieve a human-like fluency and style for the complicated task the Narrator has to perform.

H1: The score for grammar will be higher than 3 (out of 5).

For identification, I expect readers to identify more with the character from whose point of view the story is told, if focalization is enabled. I expect the identification with for example Barry, the loitering youth, to be the highest when focalization is enabled for Barry, compared to when focalization is enabled for the police officer, or when focalization is disabled.

H2a: Readers will identify more with a character when focalization is enabled for that character, compared to focalization with another character.

H2b: Readers will identify more with a character when focalization is enabled for that character, compared to when focalization is disabled.

My hypothesis for the understanding part is that most human judges will choose the event in the flashback as the cause of the action. When flashbacks are disabled, I expect they will choose some enabling event or an action that happened just before.

H3: Readers will understand the flashback to be the cause of the action in the current story.

In the case of focalization, I expect that the judges will choose some action from the police officer in the case of focalizing from their perspective, without being aware of the emotional reasoning of the adolescent, as this is not told when focalizing from the police officer's point of view. The emotional reasoning is present in both cases when focalizing from the adolescent, and when having an all-knowing narrator. In the case of an all-knowing Narrator however, there will be more information present that might distract from being aware of the emotional reasoning.

H4a: Readers will pick emotional reasoning of a character as the cause for their action, if focalization is enabled for that character, or if focalization is disabled.

H4b: Readers will pick an action as the cause for a character's action, if focalization is enabled for another character than the one that performed the action.

8.3.3 Experiment setup

For the human evaluation I created three base stories about an interaction between the loitering youths, the police officer, and bystanders. I then created different versions of these stories: without focalization, focalized from the perspective of the police officer, and focalized from the perspective of the loitering youth. I also created versions with and without flashbacks for each of these, when applicable. The flashbacks were only applicable for one character in the story, twice for the youth, once for the police. This is because the flashback is based on the emotion of a certain character. If the story is told from the perspective of another character, the emotion, and thus the flashback, is not told. This resulted in 3x5 = 15 stories. I then created six different versions of the questionnaire, so that each judge has to judge one version of each story.

In Section 8.2 I described a few things often went wrong: referring expressions, word order and the lack of ontology. To make sure these errors did not distract from the evaluation, I manually edited the stories. When a referring expression made the story hard to understand, or when the word order of a sentence was obviously wrong, I edited them. I also faked an ontology by making sure unique entities, like locations, always have a definite article. Also, in one story, the loitering youth has a moped, that is referred to as 'Barry's moped', 'his moped' or 'my moped', depending on what is appropriate. This was also manually edited. In Appendix C.2 I include all different versions of each story, including the original unmodified versions.

To evaluate the text quality I ask the human judges to rate the *Style* (is the style appropriate for a report after a simulation), *Grammar* (does the system not make grammatical mistakes) and *Fluency* (does the story feel natural, as if it is written by a human) on a 1 to 5 Likert scale.

In the second part I ask them how much they identify with the loitering youth and the police officer. This is done by a set of questions based on the work of Igartua (2010, Table 1). Four questions that seemed the most appropriate for this domain were chosen from the EDI scale. The participants were asked to answer these four questions for the loitering youth and the police officer for each story.

To measure how well people understand the text, I ask for one action each of the police officer and loitering youth what they think the cause for the action was. Causes can be an emotion like 'the police officer was angry', or an action of another character, such as 'Barry threw a can on the floor'. One of the actions for each story is one that has a causal relation with the event from the flashback, through an emotion of one of the characters. An example is: *"Richard scratched Barry's moped, therefore Barry was angry and he yelled at Richard"*. In this example, Richard scratching Barry's moped is the flashback, and the judges had to choose the cause for the action in which Barry yells at Richard. The other action of the two is one that is not caused by a past event, but by an action of one of the other characters in the current story.

In this part I also ask the judges if they thought any information was missing, and if they understood the story well, on a 1 to 5 Likert scale. In Appendix C.1, I include all questions that the judges had to answer.

8.3.4 Results

Demographics

28 people participated in the research, with ages between 16 and 71. A large majority (68%) of them was between 20 and 30 years old. The participants were recruited using social media, in particular Facebook and Tumblr. 68% of the participants identified as women, 18% identified as men, while 14% identified as neither or preferred not to disclose their gender. All participants had Dutch as their first language.

Text quality

	Style	Grammar	Fluency
Story 1	2.64	3.36	2.14
Story 2	2.75	3.32	2.46
Story 3	3.14	3.71	2.79
Total	2.67	3.38	2.23

Table 8.1: Mean values for style, grammar and fluency for the three different stories, and total.

Table 8.1 shows the mean values for style, grammar and fluency, as judged by the participants of the human evaluation. As expected, the grammar of the stories was rated much higher than the style and fluency. This is the case for all separate stories as well as the total.

Interestingly, Story 3 scored significantly higher ($p < 0.05$) than Story 1 for all three metrics. The differences between Story 1 and Story 2 were not as large, except for the Fluency, which was rated significantly higher than the Fluency of Story 1.

Identification

	Focalized Adrie	Focalized Barry	Not focalized
Adrie	2.88	2.40	2.71
Barry	1.90	2.49	2.40

Table 8.2: Means of the identification scores for each character for different focalization modes.

	Foc A	Foc B	No
Focalization A	-	**0.021**	0.149
Focalization B		-	0.065
No focalization			-

(a) p-values for the identification with Adrie, the police officer.

	Foc A	Foc B	No
Focalization A	-	$<$**0.001**	**0.004**
Focalization B		-	0.262
No focalization			-

(b) p-values for the identification with Barry, the loitering youth.

Table 8.3: p-values for single-sided paired T-test with N=28. Bold denotes $p < 0.05$

For the identification part, judges were asked four questions on a 5-point Likert scale on the identification with both Barry the loitering youth, and Adrie the police officer. The mean of these four 1-5 scores was then used, resulting in an 'identification score'. Table 8.2 shows the mean identification scores for different focalization modes, while Table 8.3 shows the p-value from a single-sided paired T-test comparing the scores for different focalization modes[1].

The identification score is influenced by focalization. Enabling focalization for a certain character did not result in a significantly higher score when compared to telling the stories without focalization, i.e. from an all-knowing narrator. However, when the story is told from the point of view of Barry, the readers identified more with Barry than in the case where the story was focalized from the point of view of Adrie. This difference was significant.

Understanding of the story

Table 8.4 shows how high the judges rated their own understanding of the story. Enabling flashbacks caused the understanding to be higher than when flashbacks were disabled, but this difference is not significant. When a flashback was not applicable to the story, because it did not apply to the focalized character, the self-reported understanding was halfway between the disabled and enabled flashbacks.

Figure 8.4 shows the causes that the human judges chose for the action which had a causal relationship with the flashback event. I grouped the results in four categories, based on what cause the judges chose. The first is the flashback event. The second is an emotion of the character that performed the action. The third is an action by some other character, and the last is the option 'Other / can't tell'.

When flashbacks were enabled, the judges chose the event from the flashback half of the time. This depends on the story: for Story 2 only 10% of the judges chose the flashback as the cause for the action, while for Story 3 all judges did. The other times they chose an emotion slightly more often than an action. Only one judge answered that they could not tell.

[1] 'Foc A' refers to stories focalized from the perspective of Adrie, the police officer, while 'Foc B' refers to stories focalized from the perspective of Barry, the loitering youth.

	avg. understanding	p
Flashback disabled	2.96	-
Flashback enabled	3.46	0.074
Flashback N/A	3.23	0.194

Table 8.4: Means of the self-reported understanding of the story, and the p-value of the single-tailed independent T-test comparing with the case of disabled flashbacks.

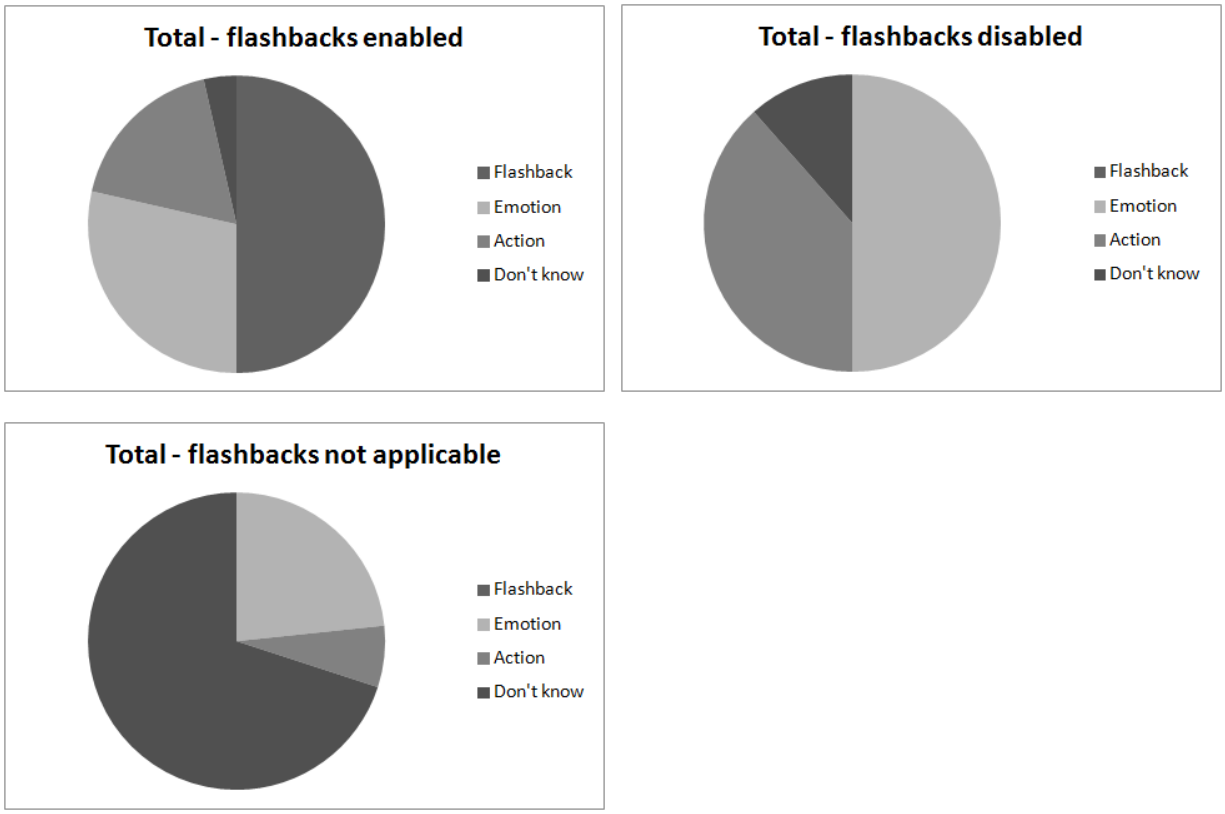

Figure 8.4: Causes chosen by the human judges for the action that was caused by the event in the flashback, grouped by flashbacks. *Flashback* means the action from the flashback, *Emotion* means an emotion of the character that performed the action, *Action* means an action by another character, and *Don't know* means the judge chose "Other or unable to tell".

When flashbacks were disabled, the judges never chose the flashback event. Again, they chose an emotion slightly more often than an action. In this case, three judges answered they could not tell. In the case of disabled flashbacks, slightly more than two thirds of the judges said they could not tell what caused the action. The other judges mostly chose an emotion as the cause.

Figure 8.5 shows the causes human judges chose for both questions in the questionnaire, this time grouped by focalization. The same categories were used as in Figure 8.4, except this time a goal of the character that performed the action was added to the "Emotion" category, as they are both cases of internal reasoning within the character. There are three categories for focalization: either the story was told by the same character as that performed the action (*"Focalization character"*), by the other character (*"Focalization other"*), or by the all-knowing narrator (*"No focalization"*).

The chosen causes look not very different between focalization by the character that performed the action, and no focalization. For the first, an action was chosen slightly more often, and for the second an emotion was chosen slightly more often. The same percentage of judges chose the flashback event.

When the story was told through another character as that performed the action, a flashback was never chosen, as it was never applicable to these stories. In this category, half of the judges could not tell what the cause of the option was. When they did choose a cause, they mostly chose an action of another character.

Open questions

During the evaluation, participants had two opportunities for giving an open answer. The first was to the question "Wat vond je van dit verhaal?" (*What did you think of this story*). In the Paragraph **Text quality** I will describe the answers to that question. The second question was "Heb je het gevoel dat er

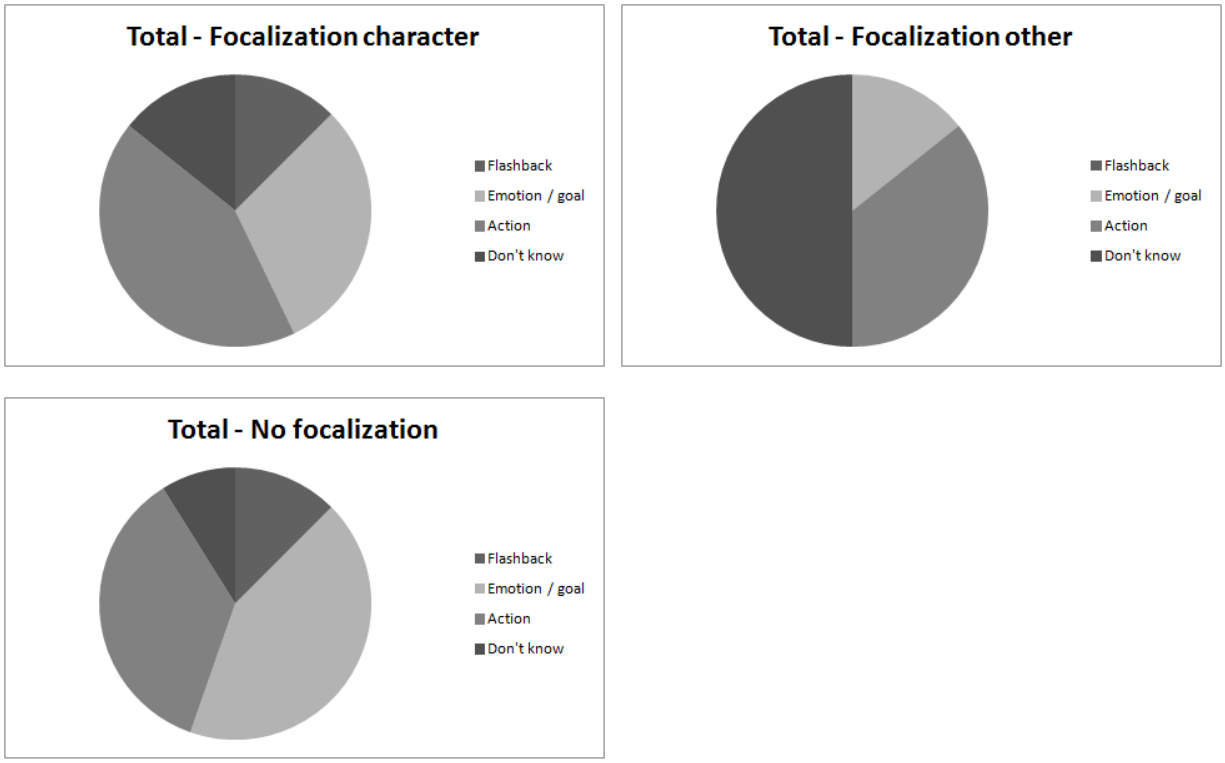

Figure 8.5: Causes chosen by the human judges for both actions in the questionnaire, grouped by focalization. *Flashback* means the action from the flashback, *Other* means some enabling action, and *Don't know* means the judge chose "Other or unable to tell".

informatie ontbreekt in dit verhaal? Zo ja, wat?" (*Do you feel like information is missing from this story? If yes, what?*). The answers to that question will be described in the Paragraph **Understanding**.

Text quality An often-heard comment from participants was that the sentences were very short. Some participants also felt like the sentences were not connected in a fluent way, resulting in a stammering text that felt more like a dry telling of facts than a story. The short sentences and the lack of connectedness of the sentences made the story look very computer-generated according to one participant.

One participant noted that some story events are told twice: once when a character performs the action, then again when another character perceives it. When focalization was enabled, some participants thought that the story contained a lot of sentences that started with 'I'.

Some participants felt that the story sounded like it was written by someone who was either very young or still learning Dutch.

Understanding When the story was told with focalization enabled, the participants often said they did not understand the causes of some actions of the other character. When flashbacks were disabled, some participants wrote that they did not understand why the effect of the flashback happened, they asked for example why Barry was angry. One participant gave examples of story events of which he did not understand the cause, such as "why did the police officer leave". In this case, the story did not tell the cause of the actions of the officer, as the story was told from the youth's perspective. A lot of participants missed information of the characters and the background of the events in the story. One participant wanted to know what the loitering youth looked like. Some readers noted that they felt like the cause of some actions was hard to understand, as the short stories lacked context. One participant did not really understand the motivation behind some of the actions of the characters. Another thought it was weird that one of the characters was called a bystander, asking what event he was a bystander of. Even though the referring expression generation was manually edited, some participants still did not understand to who some pronouns referred.

8.3.5 Discussion

Text quality The grammar score for all stories was significantly higher than the style and fluency scores, so Hypothesis H1 is verified. It is not surprising that the generated stories scored a lot higher on grammar than on fluency and style. Creating stories with correct grammar is an easier task for an NLG system: as long as the grammar rules are correctly implemented, the system should not make mistakes

with grammar. Creating a story with a good fluency and correct style is however a much more vague task, that's not easily defined with rules. It should be noted however that the grammar score could be artificially high, as word order mistakes were manually corrected.

Story 3 scored significantly higher on all three measures. There are a few possible explanations for this. The first is that the quality of the content, in the form of the Fabula, has a large impact on the overall quality of the story. It makes sense that a story with better and more interesting content will be rated higher by human judges. Another possible explanation is that the judges rated the later stories better, simply because they came later in the questionnaire.

Style A lot of participants in the human evaluation thought that the generated stories did not feel very human, and that the style was not fit for a report after a training. In the open questions, it became clear that this was in part caused by the short sentences the Narrator generates. In general, the Narrator will combine two or three clauses into a sentence. This made sense for the fairy tale domain, as fairy tales often have short sentences. For a report however longer sentences might be more appropriate.

Identification For both Adrie and Barry the identification score was the highest when the story was told through their perspective, but there was no significant difference. It seems that for these kind of stories, it matters if the reader knows the internal reasoning behind what a character does, but it does not really matter if the story is filtered to only show the actions that involve that character. This means that Hypothesis H2a is verified, but H2b is not.

It could be that the filtering does not have much impact because the stories in the questionnaire were short. It's possible that longer and more realistic stories, for example from the output of the serious game, benefit more from filtering out the actions of the other characters to make the reasoning of the focalized character more clear. A new evaluation will be needed when the serious game is finished to verify this.

Understanding There's a difference between the self-reported understanding between disabled and enabled flashbacks, but this difference is not significant with $\alpha = 0.05$. However, it does come close ($p = 0.074$), and it makes sense that removing some piece of information from the story makes the story harder to understand.

When flashbacks were enabled, the judges chose the flashback as the cause for the action half of the time. This is not bad, it shows that a lot of readers understand that the flashback caused an action in the current story. The difference between stories does however show that the effectiveness of a flashback depends on the content of the story. Since a lot of judges identified the flashback as the cause of an action, Hypothesis H3 is verified.

When flashbacks are disabled, a lot of judges answered that they could not tell what caused the action. We see the same thing when the story is focalized from another character than the one that performed the action. It makes sense that the two correlate: the case where flashbacks don't apply only occurs when the story is focalized from a character that is not part of the flashback.

The cases for focalization from the character that performed the action, and all knowing narrator, have more or less the same division of answers. Not a lot of people said they could not tell, some chose the flashback action when it was available, and the other answers are more or less evenly divided between Emotion and Action. When the story was told without focalization, more people chose an emotion of the character than when focalization was enabled for that character. Hypotheses H4a and H4b can therefore not be verified.

This follows the same pattern as the identification part of the questionnaire: the case of focalization of the character we want to identify with or understand, and the case of an all-knowing narrator do not differ much in terms of understanding and identification. However, when the story is told from the perspective of the other character, both identification and understanding suffer. This makes sense: if the story is focalized from one character, the Narrator filters out information about the internal reasoning of the other character. This causes the reader to understand that other character less, and identify less with them as well.

Perceptions The current stories often contain story events that are told twice, once as the actual event and once as a Perception event when a character perceives it. This Perception event is needed when the action of one character causes an action of another character, and the first character's action did not involve the second character. In this case, the action of the first character would not appear in the focalized story of the second character, so a Perception event is needed to make it appear anyway. The result of this is however that the event is told twice when focalization is disabled.

The solution to this would be to filter the Fabula by removing the Perception of an event if the event is already told. The current Narrator does not do this, it would be a good idea to add this in future versions of the Narrator.

Chapter 9

Conclusions and future work

In this final part I present my findings. I describe the work I have done on the Narrator, and the results of the evaluating the Narrator. Based on these results I write recommendations for future work.

9.1 Conclusions

Project goals In this report I described my work on the Narrator, a system that tells the events from the Virtual Storyteller in natural language. The Virtual Storyteller is a multi-agent system that simulates interaction between characters. Currently, a new version of the Virtual Storyteller is being developed in the form of a serious game that trains police officers for social situations.

Before this project, a version of the Narrator already existed. This system was able to convert a causal network containing a log of the events from the Virtual Storyteller to natural Dutch text. The input for this Narrator is the Fabula, a graph containing the events linked by causality, i.e. which events causes other events. The Narrator consists of three parts. A Document Planner, which performs content selection, adds background information and determines the general structure of the story. A Sentence Planner, which determines what words to use to describe each event. And finally, a Surface Realizer, which inflects words, combines sentences and adds punctuation. This system worked well, but it was made for a fairytale domain and could still be improved.

The goal of this project was to update the existing Narrator to this new serious game domain. As the serious game uses a data format that is different from that of the old Virtual Storyteller, the Narrator needed to be updated to support the new data formats. The data formats used by the Narrator were to be updated as well.

To try and improve how much the reader learns from the generated reports, two things were to be added. Focalization, i.e. telling the story from the perspective of a certain character, and flashbacks, i.e. telling events from the past. Both are designed to give the reader more insight in what caused the actions of the A.I. characters.

Focalization was added by adding a content selection step to the Document Planner that removes all events from the causal network that did not involve the character from whose perspective the story was told. Three options for perspective were added: first, second and third person. For first and second person perspectives, the appropriate pronouns were used for the focalizing character, and all verbs that applied were inflected to match this.

Flashbacks were added by using the emotions of the characters. If the causal network contains an emotional state of a character that was not caused by an event in the current story, the flashback module assumes that it was caused by an event in a previous session. It then searches a history file with the causal networks of previous sessions, and adds the last cause for that emotion for the same character to the current causal network. The flashback event is then marked, so the Narrator can later realize it in the perfect tense, and add the word ‘eerder’ (earlier).

Evaluation The system was evaluated in two ways. First, I analyzed the generated stories to see if focalization and flashbacks worked well, as well as looking for problems with language generation. I found that focalization and flashbacks worked as they should, even when used together. I did find a few language generation errors. While these fall outside of the scope of this project, it is useful to mention them for future projects on the Narrator. The Narrator has some problems with referring expression generation, it is not very good at determining when a pronoun or a description should be used. There are also a few problems with word order, modifier words like ‘niet’ (not) often end up at the wrong place in the sentence. The lack of an ontology containing extra information on entities caused the Narrator to sometimes use an indefinite article for new objects or locations when a definite article should be used, and made it impossible to communicate ownership, e.g. ‘his knife’.

The second part of the evaluation was a human evaluation, in which judges were presented with three different stories and were asked to answer questions about them on the subject of text quality, identification, and understanding. To make sure the judges were not distracted too much by obvious errors, I manually edited some stories to fix the language generation problems mentioned before, i.e. referring expressions, word order, and ontology. I found that the Narrator was rated well on grammar, but got lower ratings on style and fluency. One of the stories got significantly higher scores on all three measures, leading me to believe that the input of the Narrator plays an important role in the quality of the final text.

I found that focalization has a negative effect on identification with a character in the case where the story was told through the perspective of another character. When comparing the identification scores for a certain character, no significant difference was found between stories with an all-knowing Narrator, and stories being told through the perspective of that character. Something similar was observed for understanding of the story. Telling the story through the perspective of a certain character made understanding for the actions of the other characters worse, but did not make the understanding of the actions of that character better when compared to telling the story through an all-knowing Narrator.

This makes sense, as focalization actively removes information about other characters. I hypothesized that focalization would make the actions of the focalizing character more clear by removing other information that could be distracting, but this effect was not observed for the stories in the evaluation. This could be because they were rather short. Finally, I found that flashbacks had a positive effect on the understanding of the story, but the difference was not significant.

9.2 Suggestions for future work

I've identified four suggestions for future work that can be done to improve the stories generated by the Narrator. Firstly, an ontology can be added to include some more information on the characters and entities. The system can be evaluated using the output from the serious game once it is finished, to find out if the conclusions still stand using more realistic input. The pronominalization algorithm of the Referring Expression Generation module can be improved. And finally, content selection based on perception can be added.

Ontology The information the Narrator has on characters and entities like objects is a bit limited. Currently, the name, gender and role of characters are stored. For objects, only the word used to describes the object and the location it is in are stored. By adding properties like uniqueness, and relations between characters and objects, the Narrator can potentially create better stories.

This information can be added in the form of an ontology that describes the relations between characters, entities and concepts. Because such an ontology should be shared with the Virtual Storyteller, which is currently being redeveloped, it was not possible to create one during the course of this project. When the Virtual Storyteller adds an ontology, the Narrator should be modified to use the information contained in it.

Evaluation with serious game output The goal of this project was to create a system which automatically generates reports for a serious game about the interaction between police officers and loitering youth. However, this serious game is currently still in development. Because of that, the Narrator has been evaluated with manually written input. This has two problems: because manually generating a Fabula is a lot of work, the stories were short. The manually created stories might also not be realistic input for the task the Narrator is supposed to perform.

When the serious game is finished, it makes sense to evaluate the Narrator again, this time with the actual input from the game.

Referring Expression Generation As mentioned in Section 8.2, the Narrator still has some problems with referring expression generation. The part that causes the most problems is the pronominalization step: the algorithm that decides whether to use a pronoun such as *he*, or something else like the name of the character. Updating this algorithm can improve the quality of the generated text.

Perceptions and content selection The Virtual Storyteller, which is the basis for the serious game that generates the input for the Narrator, did not have a model for determining what the characters can and cannot perceive of the story world. Very recently, such a model was presented by ten Brinke (2014). For the stories used in the evaluation, perceptions were manually added when they were relevant.

The Narrator currently does not perform content selection on the perceptions of characters. This can lead to actions being told twice, for example *'The police officer walked to the main street. The loitering youth saw that he walked to the main street.'*. By adding a content selection module based on the perception model of ten Brinke, this can be prevented.

Appendix A

Detailed description of the Fabula Model

Overview

In this section I will describe the format I use for Fabulas in more detail. I will give an example of each type of Fabula node and edge as well.

The Fabula model follows the GraphML specification. GraphML is a standardized format for storing graphs in XML, which can be easily read by Java using existing libraries. The Fabula graph has six different types of nodes: **action**, **state**, **event**, **goal**, **perception**, and **outcome**.

Each node can have seven different types of properties: **EventType**, **Type**, **Agens**, **Patiens**, **Instrument**, **Target** and **Time**. The six types of nodes are specified as the EventType of the node. Type refers to different things based on the EventType. I will specify for each different type of node what Type refers to in their respective paragraphs.

Agens, Patiens, Instrument and Target refer to the entities involved in each story event. They are translated to the syntactic elements of subject, object and modifier respectively. Target is a bit special, in the sense that it can refer to different elements in the sentence. It can refer to a secondary object in the case of ditransitive verbs, or to a location for verbs such as 'go to'. The Time is the time stamp of the story event. It can be any positive integer up to $2^{63} - 1$ (the largest number that fits in a Java long), as long as there are no two nodes with the same time stamp. I recommend using the number of milliseconds since January 1st 1970, as this make the time values continuous between different stories, which is useful for generating flashbacks. It also greatly decreases the chance of having the same time stamp twice. The examples given use short time values for simplicity.

Basic structure

To correctly create a GraphML file, the file should link to the GraphML specification. All types of data keys, such as Type, Agents and Time also have to be specified in the file itself.

To prevent mixing up which lexicon and scenario belongs to which Fabula file, these files should be specified in the Fabula. This information is specified in XML comments, to keep the file compatible with the GraphML specification. There are three files that have to be specified: the World Setting (or scenario) file, the Lexicon, and the History, which is used for flashbacks. Specifying a History is optional, but without one flashbacks will not work. A Fabula file should always specify a Lexicon and World Setting.

An empty Fabula file, which can be used as a base for Fabulas, can be found in Figure A.1.

Action

An action is a story event performed by a certain character. Actions should always have at least a Type, Agens and Time value specified. An example of an action node can be found in Figure A.2. For action nodes, the Type refers to the type of action that is performed, for example 'fall' or 'hit'. The verb used here has to be specified in the lexicon.

An action can be either successful or unsuccessful. This is specified using the 'Successful' tag. For actions that are successful, the value does not have to be specified as it defaults to true. In Figure A.2 it is however included for clarity.

States

States, also called internal elements, reflect an emotion of a certain character. This does not necessarily have to mean a change in emotion, it can also be used when a character is already in a certain emotional

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">

  <!--World Setting file: world.xml-->
  <!--Lexicon file lang="NL": lexicon_NL.xml-->
  <!--History file: history.graphml-->

  <key id="EventType" for="node" attr.name="EventType" attr.type="string">
    <default>NoType</default></key>
  <key id="Type" for="node" attr.name="Type" attr.type="string">
    <default>NoType</default></key>
  <key id="Agens" for="node" attr.name="Agens" attr.type="string">
    <default></default></key>
  <key id="Patiens" for="node" attr.name="Patiens" attr.type="string">
    <default></default></key>
  <key id="Target" for="node" attr.name="Target" attr.type="string">
    <default></default></key>
  <key id="Instrument" for="node" attr.name="Instrument" attr.type="string">
    <default></default></key>
  <key id="Successful" for="node" attr.name="Successful" attr.type="string">
    <default>true</default></key>
  <key id="Time" for="node" attr.name="Time" attr.type="integer">
    <default>-1</default></key>
  <key id="RelType" for="edge" attr.name="RelType" attr.type="string">
    <default>NoType</default></key>

  <graph id="Fabula" edgedefault="directed">

  </graph>
</graphml>
```

Figure A.1: An empty Fabula file with scenario file *world.xml*, lexicon *lexicon_NL.xml* and history *history.graphml*.

```
<node id="Hit01">
  <data key="EventType">action</data>
  <data key="Type">hit_with</data>
  <data key="Agens">loiter1</data>
  <data key="Patiens">policeofficer</data>
  <data key="Instrument">schoen1</data>
  <data key="Successful">true</data>
  <data key="Time">10</data>
</node>
```

Figure A.2: Example of a Fabula action node.

state, but it is relevant for the story to tell it. For example because a character's emotion caused them to perform a certain action. A state node needs to have at least an Agens, a Type and a Time value. For states, the Type value refers to the emotion this character has. Each emotion has to be defined in the lexicon. An example of a state node can be found in Figure A.3.

```
<node id="BecomeAngry01">
  <data key="EventType">state</data>
  <data key="Type">angry</data>
  <data key="Agens">loiter1</data>
  <data key="Time">6</data>
</node>
```

Figure A.3: Example of a Fabula state (emotion) node.

Event

Events are things that happen in the story world. They are basically actions with the Agens being something other than a character. Examples are 'a door opens', 'an acorn fell on a character'.

An event needs at least a Type, Agens and Time value. Like an action, the Type refers to the action that is performed. An example of an event node can be found in Figure A.4.

```
<node id="Open01">
    <data key="EventType">action</data>
    <data key="Type">open</data>
    <data key="Agens">door1</data>
    <data key="Time">9</data>
</node>
```

Figure A.4: Example of a Fabula event node.

Goal

Goals are a bit more complicated than actions, states or events. Goals use a 'sub-plot element', which contains the story event the goal refers to. For example: if the police officer has a goal to speak to a certain character, the goal node has a sub-plot element containing the action of them speaking to that character. The sub-plot element node is connected to the goal node by using a special type of edge, with a RelType of **subplotelement**. The sub-node can be either an action or a state.

A goal can have several different types, specified in the EventType value. These values are **attaingoal** (character wants something to happen), **sustaingoal** (character wants something to stay happening), **leavegoal** (character wants something to stop happening) and **avoidgoal** (character wants something to not happen).

Goals need at least an Agens, Time and a sub-node telling what the goal is. An example of a goal node with a sub-plot element can be found in Figure A.5.

```
<node id="Goal01">
    <data key="EventType">attaingoal</data>
    <data key="Agens">policeofficer</data>
    <data key="Time">2</data>
</node>

<node id="Goal01sub">
    <data key="EventType">action</data>
    <data key="Type">speak_to</data>
    <data key="Agens">policeofficer</data>
    <data key="Patiens">loiter1</data>
</node>

<edge id="SubPlotElement01" source="Goal01sub" target="Goal01">
    <data key="RelType">subplotelement</data>
</edge>
```

Figure A.5: Example of a goal node using a sub plot element.

Perception

Perceptions refer to a character perceiving that another event happens. Perceptions use a sub-plot element in the same way as goals do. In the case of perceptions, the sub-node refers to the story event that is perceived.

Perceptions need at least an Agens, a Type, a Time value and a sub-node containing the perceived event. For perceptions, the Type refers to the action that is used to describe the perception, such as *see* or *hear*. As with other types of nodes, the verb has to be specified in the lexicon. An example of a perception node can be found in Figure A.6.

Edges

Edges, with the exception of the subplotelement edge defined before, refer to the causal links between story events. I.e., an edge from one node to another specifies that the first story event directly or indirectly caused the second story event. There are four types of edges: **motivates**, **psi-causes**, **phi-causes** and **enables**.

phi-causes is used to describe a physical causality. It's a strong relation between an Action and an Event, two Actions, two Events, used when something physically causes something else. An example is when a loitering adolescent kicks a garbage can, causing it to fall over. The Action 'kicks garbage can' and Event 'garbage can falls over' are then linked by a physical causality. This type of edge is also used when an Action or Event directly causes a State action, for example when one character's action causes a change of emotion in another character.

```
<node id="Perception01">
    <data key="EventType">perception</data>
    <data key="Type">see</data>
    <data key="Agens">loiter3</data>
    <data key="Time">10</data>
</node>

<node id="Perception01sub">
    <data key="EventType">action</data>
    <data key="Type">hit_with</data>
    <data key="Agens">loiter1</data>
    <data key="Patiens">policeofficer</data>
    <data key="Instrument">schoen1</data>
</node>

<edge id="SubPlotElement02" source="Perception01sub" target="Perception01">
    <data key="RelType">subplotelement</data>
</edge>
```

Figure A.6: Example of a perception node using a sub plot element.

```
<edge id="Psychological02" source="Hit01" target="BecomeSad01">
    <data key="RelType">phi−causes</data>
</edge>

<edge id="Physical04" source="BecomeAngry01" target="Kick01">
    <data key="RelType">psi−causes</data>
</edge>

<edge id="Motivation01" source="Goal01" target="Talk01">
    <data key="RelType">motivates</data>
</edge>

<edge id="Enables02" source = "Perception01" target="Talk01">
    <data key="RelType">enables</data>
</edge>
```

Figure A.7: Examples of Fabula edges

psi-causes describes a psychological causality. This is used for internal reasoning within a character. It is used when a Perception causes a State node, a State causes another State node or when a State causes a goal.

motivates describes a relation where one node motivates another. This is used when a Goal motivates another Goal, which has to be completed before the main Goal can be completed. It is also used when a Goal or State node motivates an Action, for example when a loiter runs away, because they are scared.

enables is a weak relation which is used when a node does not cause another node, but satisfies a precondition that was needed for the second node to happen. An example is when a police officer wants to talk to a loiter. The Action 'talk to loiter' is caused by police officer's goal, but it can be enabled by the police officer seeing that she is close to the loiter.

Examples of all four edge types can be found in Figure A.7. An overview of what edges should be used to connect which nodes can be found in Table A.1. Note that the enables relation is never present, because this can be used for any relation as long as it specifies an enabling relation.

	Goal	State	Action	Event	Perception
Goal	m	m	m	-	-
State	m	psi	m	-	-
Action	-	phi	phi	phi	phi
Event	-	phi	phi	phi	phi
Perception	-	psi	psi	-	-

Table A.1: Specification of what edges should connect what nodes.

Appendix B

Example story generation

In this Appendix I will show the generation of a simple story. For this example story, I chose to enable flashbacks and focalize from the perspective of the loitering youth in first person mode. I also chose to generate the text in the present tense.

Fabula The story starts with a pre-defined Fabula, which is the output from the Virtual Storyteller or the AGENT serious game. In this case I have manually created a simple Fabula with only a few events. The Fabula graph can be found in Figure B.1, the story events corresponding to each Fabula node can be found in Figure B.2. Note that subplotelements, i.e. Goal01sub and Perception01sub, are already combined with their parent elements in this list.

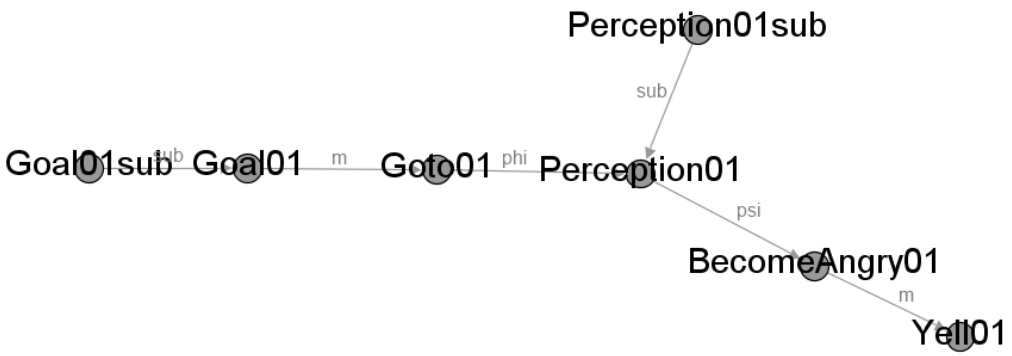

Figure B.1: The Fabula graph that is used as input to the Narrator.

Goal01 The police officer wants to talk to the loitering youth.

Goto01 The police officer goes to the main street.

Perception01 The loitering youth sees that the police officer goes to the main street.

BecomeAngry01 The loitering youth is angry.

Yell01 The loitering youth yells at the police officer.

Figure B.2: The plot elements for the Fabula in Figure B.1.

Content selection (Focalization) The Narrator starts by reading the GraphML file. If a story node contains a subplotelement, the data of the subplotelement is added to the main element, and the subplotelement node is removed. If focalization is enabled, it will first filter this graph by removing all events that the focalized character does not participate in. In this case, the event in which the police officer goes to the main street is removed, as well as the police officer's goal.

Content addition (Flashbacks) If flashbacks are enabled, the Narrator will search a history Fabula for events that created a certain emotion in a character. In this case, the history file contains a sequence of events in which the police officer made the loitering youth angry by insulting him. This matches the event in the current story in which Barry is angry. The Flashback module will therefore add the event

in which the police officer insulted Barry, and mark it as a flashback. In Figure B.3, the new Fabula file created after these two steps is shown.

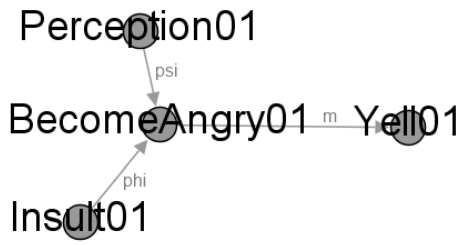

Figure B.3: The Fabula graph after performing focalization and flashbacks.

Document Planning The next step is to create a document plan from this new Fabula. This is done by an algorithm that converts the Fabula graph to a binary tree of the relations between story events. Each Fabula node is also converted to a new type of object, the Plot Element. The resulting document plan as created by the Narrator can be found in Figure B.4 [1].

Normally the document planner would also add an introduction for the characters, and add the names of each character in an elaboration relation the first time they are mentioned. However, to keep the document plan simple I have disabled these options.

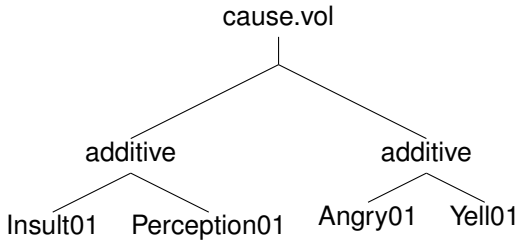

Figure B.4: The document plan created from the Fabula.

Sentence Planning The next step is to convert the Plot Elements created in the last step to sentence plans. This is done by filling in a template for each type of plot element, e.g. action, emotional state, perception. The resulting sentence plans for each sentence in the story can be found in Figure B.5.

The sentence planner also stores for each verb how it should be inflected. For the sentences with the loitering youth as the subject, the first person should be used because the story is focalized from the loitering youth in a first person perspective. The verbs in sentences with someone else, in this case the police officer, should be inflected in third person.

The sentence with the flashback should be inflected in the perfect tense. For this sentence, the sentence planner also adds the adverb ‘earlier’ and the auxiliary verb ‘to have’. The other verbs are inflected in the present tense, because the present tense option was selected. Note that the sentence planner does not actually inflect verbs, it instead marks how they should be inflected.

Referring Expression Generation Referring expression generation refers to creating the piece of text in the sentence that describes a character or concept. For the Narrator, this means converting each instance of a concept to a referring expression.

Because the story is focalized in the first person view of Barry, the loitering youth, all instances of ‘loiter1’, i.e. the concept referring to Barry, are converted to ‘ik’ (I) when Barry is the subject, or ‘mij’ (me) when Barry is the primary or secondary object of the sentence.

The police officer can be referred to with a pronoun, i.e. ‘hij’ (him), his name, i.e. Adrie, or his role, i.e. ‘politieman’ (police officer). In this story, the referring expression generator chose ’politieman’ three times. The first time, the indefinite article ‘een’ (a) was used, because there might be more than one police officer. The next two times, a definite article is used because the reader now understands ‘politieman’ refers to the same police officer as before.

The main street does not have a name, and pronouns do not apply to it, so the referring expression generation can only choose ‘hoofdstraat’ (main street). Like before, an indefinite article is chosen because this is the first time the main street is referred to.[2]

[1] Note that the document planning is not ideal in this example. The relations between the sentences are planned as additive instead of as causes, making it not entirely clear what caused some actions.

[2] In this case, it would make more sense to use the definite article right away, because there is only one main street. However, the Narrator currently has no way of distinguishing between unique and non-unique entities.

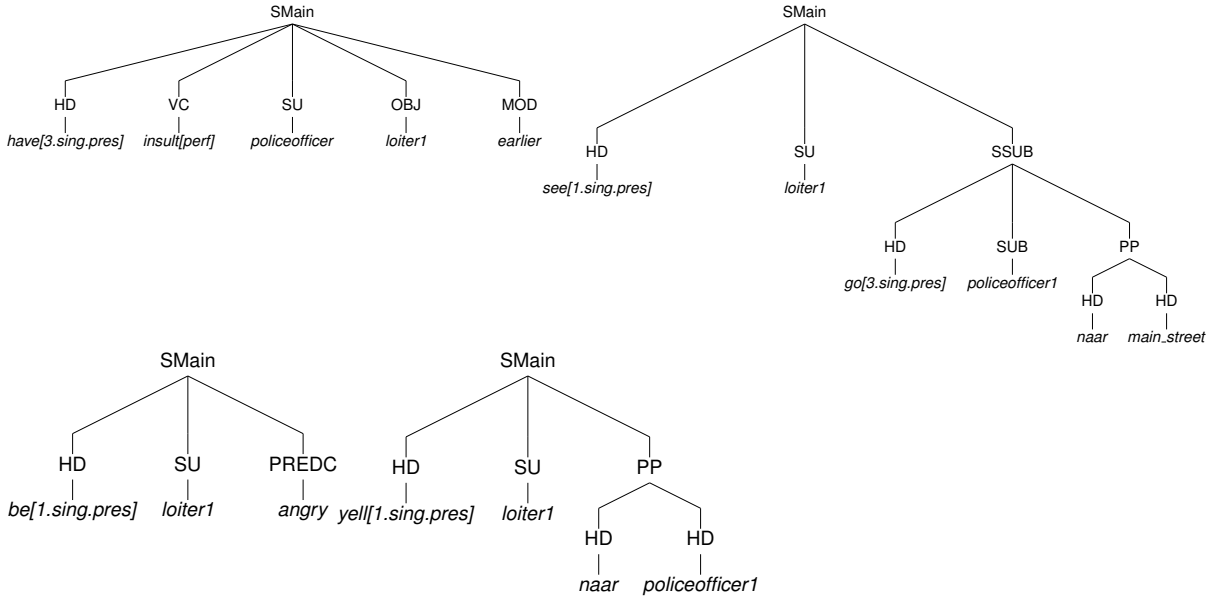

Figure B.5: The created sentence plans for each sentence in the story.

S1	policeofficer: “een politieman”	loiter1: “mij”	
S2	loiter1: “ik”	policeofficer: “de politieman”	main_street: “een hoofdstraat”
S3	loiter1: “ik”		
S4	loiter1: “ik”	policeofficer: “de politieman”	

Sentence Aggregation In this step, two or more sentences are combined into one sentence. To do this, the sentence aggregator uses the relations from the document plan and converts them into cue words. These cue words are used in the surface text to communicate to the reader what the relation between the combined sentences is.

In this case, sentence aggregation is relatively simple: there are two sets of two sentences each connected by an additive relation in the document plan. The sentence aggregator thus combines sentences 1 and 2, and sentences 3 and 4, and uses the cue word ‘en’ (and) for both new sentences.

Creation of surface text To create the surface text, the sentence plans are put in order using a template for each combination of constituents. The verbs, adjectives and nouns are inflected to their surface forms, and finally punctuation is added. This results in the surface text of Figure B.6.

```
Eerder heeft een politieman mij beledigd en
ik zie dat de politieman naar een hoofdstraat gaat.
Ik ben boos en ik schreeuw naar de politieman.
```

Figure B.6: The surface text of the example story.

Appendix C

Questionnaire

This appendix contains the details of the questionnaire that was used to evaluate the Narrator. It is split in two parts: the first presents the questions the judges had to answer on the generated stories, the second part presents these automatically generated stories.

C.1 Questions

In this section I present the questions that the human judges had to answer in the questionnaire. Each question is mentioned twice, once in the original Dutch form as it was present in the real questionnaire, and once translated to English.

Demographics

Wat is je geslacht?
Kies de optie waar je je het meest mee identificeert.
Opties: Man, Vrouw, Anders, Wil ik niet zeggen.

What is your gender?
Choose the option with which you identify the most.
Options: Male, Female, Other, Prefer not to say.

Wat is je leeftijd?
What is your age?

Is Nederlands je moedertaal? [Ja/Nee]
Is Dutch your first language? [Yes/No]

Text quality

Unless otherwise specified, these questions had to be answered on a 1 tot 5 Likert scale.

Wat vind je van de stijl van deze tekst?
Klopt het met wat je verwacht voor een rapport na een training?
What do you think of the style of this text?
Does it match with what you expect for a training report?

Hoe goed klopt de grammatica van deze tekst?
How correct is the grammar for this text?

Hoe vloeiend is deze tekst?
Leest het alsof het door een mens is geschreven?
How fluent is this text?
Does it read as if it were written by a human?

Wat vond je van dit verhaal? [Open antwoord]
What did you think of this story? [Open answer]

Identification

Unless otherwise specified, these questions had to be answered on a 1 tot 5 Likert scale.

Ik kan me identificeren met de hangjongere
I identified with the loitering youth

Ik had het gevoel dat ik het verhaal van de hangjongere echt ervaarde
I had the impression that I was really experiencing the story of the loitering youth

Ik begreep de manier waarop de hangjongere zich gedroeg, dacht, of voelde
I understood the loitering youth's way of acting, thinking or feeling

Ik stelde me voor hoe ik me zou gedragen in de plaats van de hangjongere
I imagined how I would act if I found myself in the place of the loitering youth

Ik kan me identificeren met de agent
I identified with the police officer

Ik had het gevoel dat ik het verhaal van de agent echt ervaarde
I had the impression that I was really experiencing the story of the police officer

Ik begreep de manier waarop de agent zich gedroeg, dacht, of voelde
I understood the police officer's way of acting, thinking or feeling

Ik stelde me voor hoe ik me zou gedragen in de plaats van de agent
I imagined how I would act if I found myself in the place of the police officer

Understanding

For Story 1

Waarom schreeuwt Barry naar de politieagent?

- Hij was boos
- De politieagent had hem beledigd
- De politieagent was in de hoofdstraat
- Barry luisterde naar zijn iPod
- Anders / niet te zeggen

Why is Barry yelling at the police officer?

- *He was angry*
- *The police officer had insulted him*
- *The police officer was in the main street*
- *Barry was listening to his iPod*
- *Other / Can't tell*

Waarom gaat de politieman naar de hoofdstraat?

- Om beledigd te worden door Barry
- Omdat Barry in de hoofdstraat was
- Omdat Barry boos was
- Om met Barry te spreken
- Anders / niet te zeggen

Why did the police officer go to the main street?

- *To be insulted by Barry*
- *Because Barry was in the main street*
- *Because Barry was angry*
- *To talk to Barry*
- *Other / Can't tell*

For Story 2

Waarom slaat Barry de politieagent?

- Hij was boos
- De politieagent had hem beledigd
- De politieagent was in de hoofdstraat
- Manuela was in de hoofdstraat
- Anders / niet te zeggen

Why did Barry hit the police officer?

- *He was angry*
- *The police officer had insulted him*
- *The police officer was in the main street*
- *Manuela was in the main street*
- *Other / Can't tell*

Waarom gaat de politieman naar de hoofdstraat?

- Omdat hij waakzaam was
- Omdat Manuela hem gewaarschuwd had
- Omdat Manuela bang was
- Omdat Manuela in de hoofdstraat was
- Anders / niet te zeggen

Why did the police officer go to the main street?

- *Because he was vigilant*
- *Because Manuela warned him*
- *Because Manuela was scared*
- *Because Manuela was in the main street*
- *Other / Can't tell*

For Story 3

Waarom schreeuwt Barry naar Richard?

- Richard was in de hoofdstraat
- De agent was in de hoofdstraat
- Barry was boos
- Richard had Barry's scooter bekrast
- Anders / niet te zeggen

Why did Barry yell at Richard?

- *Richard was in the main street*
- *The police officer was in the main street*
- *Barry was angry*
- *Richard scratched Barry's moped*
- *Other / Can't tell*

Waarom gebaart de agent naar Barry?

- Barry schreeuwde naar Richard
- De agent was blij
- Barry was boos
- De agent praatte met Barry
- Anders / niet te zeggen

Why did the police officer gesture at Barry?

- *Barry yelled at Richard*
- *The police officer was happy*
- *Barry was angry*
- *The police officer talked to Barry*
- *Other / Can't tell.*

For all stories

Ik begrijp wat er in dit verhaal is beschreven [1-5 Likert scale]
I understand what was described in this story [1-5 Likert scale]

Heb je het gevoel dat er informatie ontbreekt in dit verhaal? Zo ja, wat? [Open antwoord]
Do you feel like information is missing from this story? If yes, what? [Open answer]

C.2 Stories

In this section I present the stories as present in the questionnaire. Each of the six versions of the questionnaire had one variant of each of the three stories. I present each variant twice: once in the modified form as it appeared in the questionnaire, and once in the original form as generated by the Narrator. Difference are marked in bold in the original story.

C.2.1 Story 1

No focalization, flashbacks enabled

Modified:
Een stoere hangjongere, die Barry heette, was in de hoofdstraat.
Een politieman, die Adrie heette, was op het plein.
Hij wilde met Barry praten. Hij ging naar de hoofdstraat.
Barry luisterde naar zijn iPod en de politieman zwaaide naar hem.
Hij zag dat de politieman naar de hoofdstraat ging.
Eerder had de politieman hem beledigd, dus Barry was boos.
Hij schreeuwde naar de politieman, zodat hij ook naar hem schreeuwde.
De hangjongere pakte een mes, dus de politieman zag dat hij het mes pakte.
Omdat de politieman bang was, ging hij naar het politiebureau.
Barry zag dat de politieman weg ging.

Original:
Een stoere hangjongere, die Barry heette, was in **een** hoofdstraat.
Een politieman, die Adrie heette, was op **een** plein.
Hij wilde met Barry praten. Hij ging naar de hoofdstraat.
Barry luisterde naar **Barry's iPod** en **hij** zwaaide naar **Barry**.
Hij zag dat **Adrie** naar de hoofdstraat ging.
Eerder had **hij** hem beledigd, dus Barry was boos.
Hij schreeuwde naar **hem**, zodat hij naar hem **ook** schreeuwde.
Hij pakte een mes, dus **Adrie** zag dat hij het mes pakte.
Omdat de politieman bang was, ging hij naar **een** politiebureau.
Barry zag dat **Adrie** weg ging.

No focalization, flashbacks disabled

Modified:
Een stoere hangjongere, die Barry heette, was in de hoofdstraat.
Een politieman, die Adrie heette, was op het plein.
Hij wilde met Barry praten, dus hij ging naar de hoofdstraat.
Hij zwaaide naar hem en de hangjongere luisterde naar zijn iPod.
Hij zag dat de politieman naar de hoofdstraat ging.
Barry schreeuwde naar hem, omdat hij boos was.
De politieman schreeuwde ook naar hem en daarom pakte de hangjongere een mes.
De politieman zag dat de stoere hangjongere het mes pakte, dus de politieman was bang.
Hij ging naar het politiebureau en de hangjongere zag dat de politieman weg ging.

Original:
Een stoere hangjongere, die Barry heette, was in **een** hoofdstraat.
Een politieman, die Adrie heette, was op **een** plein.
Hij wilde met Barry praten, dus hij ging naar de hoofdstraat.
Hij zwaaide naar hem en **Barry** luisterde naar **Barry's** iPod.
Hij zag dat **Adrie** naar de hoofdstraat ging.
De hangjongere schreeuwde naar hem, omdat hij boos was.
De politieman schreeuwde naar hem **ook** en **hij** pakte een mes **daarom**.
Adrie zag dat de stoere hangjongere het mes pakte, dus de politieman was bang.
Hij ging naar **een** politiebureau en de hangjongere zag dat **Adrie** weg ging.

Focalization Adrie, flashbacks not applicable

Modified:
Een stoere hangjongere, die Barry heette, was in de hoofdstraat.
Ik was op het plein.
Ik wilde met Barry praten, dus ik ging naar de hoofdstraat.
Ik zwaaide naar hem en de hangjongere schreeuwde naar mij, zodat ik ook naar hem schreeuwde.
Ik zag dat de stoere hangjongere een mes pakte.
Daarom was ik bang. Ik ging naar het politiebureau.

Original:
Een stoere hangjongere, die Barry heette, was in **een** hoofdstraat.
Ik was op **een plein**.
Ik wilde met Barry praten, dus ik ging naar de hoofdstraat.
Ik zwaaide naar hem en **Barry** schreeuwde naar mij, zodat ik naar hem **ook** schreeuwde.
Ik zag dat de stoere hangjongere een mes pakte.
Daarom was ik bang. Ik ging naar **een** politiebureau.

Focalization Barry, flashbacks enabled

Modified:
Ik was in de hoofdstraat.
Een politieman, die Adrie heette, was op het plein.
Hij zwaaide naar mij en ik luisterde naar mijn iPod.
Ik zag dat de politieman naar de hoofdstraat ging.
Eerder had hij mij beledigd, dus ik was boos.
Ik schreeuwde naar hem, zodat hij ook naar mij schreeuwde.
Ik pakte een mes en ik zag dat de politieman weg ging.

Original:
Ik was in **een** hoofdstraat.
Een politieman, die Adrie heette, was op **een** plein.
De politieman zwaaide naar mij en ik luisterde naar **Barry's** iPod.
Ik zag dat de politieman naar de hoofdstraat ging.
Eerder had hij mij beledigd, dus ik was boos.
Ik schreeuwde naar hem, zodat hij naar mij **ook** schreeuwde.
Ik pakte een mes en ik zag dat de politieman weg ging.

Focalization Barry, flashbacks disabled

Modified:
Ik was in de hoofdstraat.
Een politieman, die Adrie heette, was op het plein.
Ik luisterde naar mijn iPod en de politieman zwaaide naar mij.
Ik zag dat Adrie naar de hoofdstraat ging.
Ik was boos, dus ik schreeuwde naar de politieman.
Ik pakte een mes, omdat de politieman ook naar mij schreeuwde.
Ik zag dat de politieman weg ging.

Original:
Ik was in **een** hoofdstraat.
Een politieman, die Adrie heette, was op **een** plein.
Ik luisterde naar **Barry's** iPod en **hij** zwaaide naar mij.
Ik zag dat **de politieman** naar de hoofdstraat ging.
Ik was boos, dus ik schreeuwde naar de politieman.
Ik pakte een mes, omdat de politieman naar mij **ook** schreeuwde.
Ik zag dat de politieman weg ging.

C.2.2 Story 2

No focalization, flashbacks enabled

Modified:
Een mooie omstander, die Manuela heette, was in de hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Een politieman, die Adrie heette, was op het plein.
Manuela was bang. Adrie hoorde dat zij bang was.
Eerder had zij hem gewaarschuwd, dus hij was waakzaam.
Adrie groette Barry, nadat hij naar de hoofdstraat ging.
De hangjongere zag dat de politieman naar de hoofdstraat ging.
De politieman beledigde hem, zodat hij boos was.
Hij sloeg de politieman met een schoen.
Doordat de politieman verdrietig was, ging hij naar het politiebureau.
De omstander, die bang was, zag dat Barry hem sloeg.
Ook zag de hangjongere dat de politieman weg ging.

Original:
Een mooie omstander, die Manuela heette, was in **een** hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Een politieman, die Adrie heette, was op **een** plein.
Manuela was bang. Adrie hoorde dat zij bang was.
Eerder had zij hem gewaarschuwd, dus hij was waakzaam.
Adrie groette Barry, nadat hij naar de hoofdstraat ging.
De hangjongere zag dat de politieman naar de hoofdstraat ging.
Hij beledigde hem, zodat hij boos was.
Hij sloeg **hem** met een schoen.
Doordat de politieman verdrietig was, ging hij naar **een** politiebureau.
De omstander, die bang was, zag dat Barry hem sloeg.
Ook zag de hangjongere dat de politieman weg ging.

No focalization, flashbacks disabled

Modified:
Een mooie omstander, die Manuela heette, was in de hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Een politieman, die Adrie heette, was op het plein.
Manuela was bang. Adrie hoorde dat zij bang was.
Hij was waakzaam, dus hij ging naar de hoofdstraat.
Barry zag dat de politieman naar de hoofdstraat ging.
Hij beledigde hem, omdat hij hem groette.
De hangjongere was boos en daarom sloeg hij de politieman met een schoen.
Manuela zag dat hij hem sloeg.
Adrie was verdrietig, zodat hij naar het politiebureau ging.

Barry zag dat de politieman weg ging.

Original:
Een mooie omstander, die Manuela heette, was in **een** hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Een politieman, die Adrie heette, was op **een** plein.
Manuela was bang. Adrie hoorde dat zij bang was.
Hij was waakzaam, dus hij ging naar de hoofdstraat. Barry zag dat de politieman naar de hoofdstraat ging.
Hij beledigde hem, omdat hij hem groette.
Barry was boos en met een schoen sloeg hij de politieman **daarom**.
Manuela zag dat hij hem sloeg.
De politieman was verdrietig, zodat hij naar **een** politiebureau ging.
Barry zag dat **Adrie** weg ging.

Focalization Adrie, flashbacks enabled

Modified:
Een mooie omstander, die Manuela heette, was in de hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Ik was op het plein.
Eerder had Manuela mij gewaarschuwd, dus ik was waakzaam.
Ik hoorde dat zij bang was.
Vervolgens ging ik naar de hoofdstraat.
Ik beledigde Barry, omdat ik hem groette.
Met een schoen sloeg de hangjongere mij, zodat ik verdrietig was, en ik ging naar het politiebureau.

Original:
Een mooie omstander, die Manuela heette, was in **een** hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Ik was op **een** plein.
Eerder had Manuela mij gewaarschuwd, dus ik was waakzaam.
Ik hoorde dat zij bang was.
Vervolgens ging ik naar de hoofdstraat.
Ik beledigde Barry, omdat ik hem groette.
Met een schoen sloeg de hangjongere mij, zodat ik verdrietig was, en ik ging naar **een** politiebureau.

Focalization Adrie, flashbacks disabled

Modified:
Een mooie omstander, die Manuela heette, was in de hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Ik was op het plein.
Ik hoorde dat Manuela bang was, en ik was waakzaam.
Ik ging naar de hoofdstraat en ik groette Barry, dus ik beledigde hem.
Met een schoen sloeg de hangjongere mij, zodat ik verdrietig was, en ik ging naar het politiebureau.

Original:
Een mooie omstander, die Manuela heette, was in **een** hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Ik was op **een** plein.
Ik hoorde dat Manuela bang was, en ik was waakzaam.
Ik ging naar de hoofdstraat en ik groette Barry, dus ik beledigde hem.
Met een schoen sloeg **Barry** mij, zodat ik verdrietig was, en ik ging naar **een** politiebureau.

Focalization Barry, flashbacks not applicable

Modified:
Een mooie omstander, die Manuela heette, was in de hoofdstraat.
Ook was ik in de hoofdstraat.
Een politieman, die Adrie heette, was op het plein.
Ik zag dat de politieman naar de hoofdstraat ging.
Hij groette mij, dus hij beledigde mij.

Omdat ik boos was, sloeg ik hem met een schoen.
Ik zag dat Adrie weg ging.

Original:
Een mooie omstander, die Manuela heette, was in **een** hoofdstraat.
Ook was ik in de hoofdstraat.
Een politieman, die Adrie heette, was op **een** plein.
Ik zag dat Adrie naar de hoofdstraat ging.
Hij groette mij, dus hij beledigde mij.
Omdat ik boos was, sloeg ik hem met een schoen.
Ik zag dat Adrie weg ging.

Story 3

No focalization, flashbacks enabled

Modified:
Een lange omstander, die Richard heette, was in de hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Ook was een politieman, die Adrie heette, in de hoofdstraat.
Eerder had Richard Barry's scooter gekrast, dus Barry was boos.
Hij schreeuwde naar de lange omstander.
Adrie hoorde dat Barry naar de omstander schreeuwde.
Nadat Adrie naar Barry zwaaide, gebaarde hij kalmerend naar hem.
Hij praatte met de hangjongere, zodat hij rustig was.
De politieman zag dat de hangjongere rustig was.
Daarom was Adrie blij.

Original:
Een lange omstander, die Richard heette, was in **een** hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Ook was een politieman, die Adrie heette, in de hoofdstraat.
Eerder had Richard Barry's scooter gekrast, dus Barry was boos.
Hij schreeuwde naar de lange omstander.
Adrie hoorde dat de **hangjongere** naar de omstander schreeuwde.
Nadat Adrie naar Barry zwaaide, gebaarde hij naar hem **kalmerend**.
Hij praatte met **hem**, zodat hij rustig was.
De politieman zag dat de hangjongere rustig was.
Daarom was **de politieman** blij.

No focalization, flashbacks disabled

Modified:
Een lange omstander, die Richard heette, was in de hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Ook was een politieman, die Adrie heette, in de hoofdstraat.
Barry was boos, dus hij schreeuwde naar Richard.
Adrie hoorde dat Barry naar de omstander schreeuwde.
De politieman zwaaide daarom naar de hangjongere.
Adrie praatte met Barry, nadat hij naar de hangjongere kalmerend gebaarde.
De hangjongere was rustig.
Adrie zag dat de hangjongere rustig was. Dus was de politieman blij.

Original:
Een lange omstander, die Richard heette, was in **een** hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Ook was een politieman, die Adrie heette, in de hoofdstraat.
Barry was boos, dus hij schreeuwde naar Richard.
Adrie hoorde dat **de hangjongere** naar de omstander schreeuwde.
De politieman zwaaide naar **hem daarom**.
Adrie praatte met Barry, nadat hij naar de hangjongere kalmerend gebaarde.
De hangjongere was rustig.
Adrie zag dat de hangjongere rustig was. Dus was **Adrie** blij.

Focalization Adrie, flashbacks not applicable

Modified:
Een lange omstander, die Richard heette, was in de hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Ik was ook in de hoofdstraat.
Ik hoorde dat Barry naar Richard schreeuwde, dus ik zwaaide naar de hangjongere.
Ik praatte met Barry, nadat ik naar de stoere hangjongere kalmerend gebaarde.
Ik zag dat de hangjongere rustig was. Daarom was ik blij.

Original:
Een lange omstander, die Richard heette, was in **een** hoofdstraat.
Ook was een stoere hangjongere, die Barry heette, in de hoofdstraat.
Ook was ik in de hoofdstraat.
Ik hoorde dat Barry naar Richard schreeuwde, dus ik zwaaide naar de hangjongere.
Ik praatte met Barry, nadat ik naar de stoere hangjongere kalmerend gebaarde.
Ik zag dat de hangjongere rustig was. Daarom was ik blij.

Focalization Barry, flashbacks enabled

Modified:
Een lange omstander, die Richard heette, was in de hoofdstraat.
Ook was ik in de hoofdstraat.
Ook was een politieman, die Adrie heette, in de hoofdstraat.
Eerder had Richard mijn scooter gekrast, dus ik was boos en ik schreeuwde naar hem.
Nadat Adrie naar mij zwaaide, gebaarde hij kalmerend naar mij.
Hij praatte met mij, zodat ik rustig was.

Original:
Een lange omstander, die Richard heette, was in **een** hoofdstraat. Ook was ik in de hoofdstraat.
Ook was een politieman, die Adrie heette, in de hoofdstraat.
Eerder had Richard **Barry's** scooter gekrast, dus ik was boos en ik schreeuwde naar hem.
Nadat Adrie naar mij zwaaide, gebaarde hij naar mij **kalmerend**.
Hij praatte met mij, zodat ik rustig was.

Focalization Barry, flashbacks disabled

Modified:
Een lange omstander, die Richard heette, was in de hoofdstraat.
Ook was ik in de hoofdstraat.
Ook was een politieman, die Adrie heette, in de hoofdstraat.
Ik was boos, dus ik schreeuwde naar Richard.
Nadat Adrie naar mij zwaaide, gebaarde hij kalmerend naar mij.
Hij praatte met mij, zodat ik rustig was.

Original:
Een lange omstander, die Richard heette, was in **een** hoofdstraat.
Ook was ik in de hoofdstraat.
Ook was een politieman, die Adrie heette, in de hoofdstraat.
Ik was boos, dus ik schreeuwde naar Richard.
Nadat Adrie naar mij zwaaide, gebaarde hij naar mij **kalmerend**.
Hij praatte met mij, zodat ik rustig was.

Bibliography

Thomas Altman. *De bruid (original: True Bride)*. Trans. Annemarie Lodewijk, 1984.

United States Army. Field manual 25-101: Battle focused training. *Department of the Army. Washington D.C.*, 1990.

Byung-Chull Bae and R Michael Young. A use of flashback and foreshadowing for surprise arousal in narrative using a plan-based approach. In *Interactive Storytelling*, pages 156–167. 2008.

Mieke Bal. *Narratology: Introduction to the Theory of Narrative*. University of Toronto, 1997.

Regina Barzilay and Mirella Lapata. Modeling local coherence: an entity-based approach. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 141–148, Stroudsburg, PA, USA, 2005.

Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks, 2009.

John A. Bateman. Enabling technology for multilingual natural language generation: the KPML development environment. *Natural Language Engineering*, 3(1):15–55, March 1997.

Christian Bizer. The TriG syntax, July 2007. URL http://wifo5-03.informatik.uni-mannheim.de/bizer/trig/.

Lynne Cahill, John Carroll, Roger Evans, Daniel Paiva, Richard Power, Donia Scott, and Kees van Deemter. From rags to riches: exploiting the potential of a flexible generation architecture. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ACL '01, pages 106–113, Stroudsburg, PA, USA, 2001.

Charles B. Callaway and James C. Lester. Narrative prose generation. *Artificial Intelligence*, 139(2):213 – 252, 2002.

Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs. *Web Semantics*, 3 (4):247–267, December 2005.

David B. Christian and R. Michael Young. Comparing cognitive and computational models of narrative structure. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, pages 385–390. AAAI Press, 2004.

Mark G Core, H Chad Lane, Michael Van Lent, Dave Gomboc, Steve Solomon, and Milton Rosenberg. Building explainable artificial intelligence systems. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1766. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

Robert Dale and Chris Mellish. Towards evaluation in natural language generation. In *Proceedings of the First International Conference on Language Resources and Evaluation*, volume 562, 1998.

Daniël De Kok and Gertjan van Noord. A sentence generator for Dutch. In *Proceedings of the 20th Meeting of Computational Linguistics in the Netherlands*, pages 75–90. University of Groningen, 2010.

Guillermo del Toro and Travis Beacham. Pacific rim, 2013. URL http://www.imdb.com/title/tt1663662/.

Michael Elhadad. *Using argumentation to control lexical choice: a functional unification implementation*. PhD thesis, Columbia University, 1993.

Andrea Falcon. Aristotle on causality. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2014 edition, 2012.

Albert Gatt and Ehud Reiter. SimpleNLG: A realisation engine for practical applications. In *ENLG-2009*, pages 90–93, 2009.

Guido Geerts, Walter Haeseryn, J. de Rooij, and M. van der Toorn. *Algemene Nederlandse Spraakkunst*. Wolters-Noordhoff, Groningen and Wolters, Leuven, 1984.

Gérard. Genette and Jane E. Lewin. *Narrative Discourse: An Essay in Method*. Cornell paperbacks. Cornell University Press, 1983.

Pablo Gervás. Stories from games: Content and focalization selection in narrative composition. In *I Spanish Symposium on Entertainment Computing*, Universidad Complutense de Madrid, Madrid, Spain, 09/2013 2013.

Pablo Gervás, Birte Lönneker-Rodman, Jan Christoph Meister, and Federico Peinado. Narrative models: Narratology meets artificial intelligence. In *International Conference on Language Resources and Evaluation. Satellite Workshop: Toward Computational Models of Literary Analysis*, pages 44–51, 2006.

Nicole Grégoire. Report on integrating ECM lexical database in Alpino system. 2007.

Brian Guarraci. SimpleNLG default lexicon, November 2011. URL https://github.com/briangu/simplenlg/blob/master/res/default-lexicon.xml.

M. Harbers, K. van den Bosch, and J.-J. Meyer. Design and evaluation of explainable bdi agents. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 2, pages 125–132, Aug 2010.

John R. Hayes and Linda S. Flower. Writing Research and the Writer. *American Psychologist*, 41(10): 1106–1113, 1986.

Feikje Hielkema. Performing syntactic aggregation using discourse structures. MSc thesis, University of Groningen, Groningen, The Netherlands, 2005. URL http://wwwhome.cs.utwente.nl/~theune/VS/FeikjeHielkema_verslag.pdf.

Heleen Hoekstra, Michael Moortgat, Bram Renmans, and Machteld Schouppe. Cgn syntactische annotatie. 2003.

Ales Horák, Piek Vossen, and Adam Rambousek. A distributed database system for developing ontological and lexical resources in harmony. In *Computational Linguistics and Intelligent Text Processing*, volume 4919 of *Lecture Notes in Computer Science*, pages 1–15. 2008.

J. J. Igartua and D. Páez. Validez y fiabilidad de una escala de empatía e identificación con los personajes. *Psicothema*, 10(2):423–436, 1998.

Juan-José Igartua. Identification with characters and narrative persuasion through fictional feature films. *Communications*, 35(4):347–373, 2010.

Emiel Krahmer and Mariët Theune. Efficient context-sensitive generation of referring expressions. *Information sharing: Reference and presupposition in language generation and interpretation*, 143: 223–263, 2002.

Andrew Lang. Andrew lang's fairy books, 1889-1910. URL http://www.mythfolklore.net/andrewlang/.

Benoit Lavoie and Owen Rambow. A fast and portable realizer for text generation systems. In *Proceedings of the fifth conference on Applied natural language processing*, ANLC '97, pages 265–268, Stroudsburg, PA, USA, 1997.

Jeroen Linssen and Thomas de Groot. AGENT: Awareness game environment for natural training. In *Proceedings of the International Conference on the Foundations of Digital Games*, FDG '13, pages 433–434, 2013.

Hugo Liu and Push Singh. Makebelieve: using commonsense knowledge to generate stories. In *Eighteenth national conference on Artificial intelligence*, pages 957–958, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

William C Mann and Sandra A Thompson. Rhetorical structure theory: A theory of text organization. Technical report, DTIC Document, 1987.

Neil McIntyre and Mirella Lapata. Learning to tell tales: a data-driven approach to story generation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 217–225, 2009.

Igor Aleksandrovič Melčuk. *Dependency syntax: theory and practice*. State University Press of New York, 1988.

Chris Mellish, Donia Scott, Lynne Cahill, Roger Evans, Daniel Paiva, and Mike Reape. A reference architecture for natural language generation systems. *Natural Language Engineering*, 12:2006, 2006.

Nick Montfort. Ordering events in interactive fiction narratives. In *Intelligent Narrative Technologies, Papers from the 2007 AAAI Fall Symposium*, pages 87–94, 2007.

Nick Montfort. Curveship's automatic narrative style. In *Proceedings of the International Conference on Foundations of Digital Games*, FDG '11, pages 211–218, 2011.

Burkhard Niederhoff. The living handbook of narratology - focalization, August 2011. URL http://wikis.sub.uni-hamburg.de/lhn/index.php/Focalization.

Steve Northover and Mike Wilson. *SWT: The Standard Widget Toolkit, Volume 1*. Addison-Wesley Professional, first edition, 2004.

NHJ Oostdijk. Het Corpus Gesproken Nederlands. 2002.

Oracle. SAXParser (Java Platform SE 7), 2013. URL http://docs.oracle.com/javase/7/docs/api/javax/xml/parsers/SAXParser.html.

Gerald Prince. *A dictionary of narratology*. University of Nebraska Press, 2003.

Ehud Reiter and Anja Belz. An investigation into the validity of some metrics for automatically evaluating natural language generation systems. *Computational Linguistics*, 35(4):529–558, 2009.

Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems*. Studies in Natural Language Processing. Cambridge University Press, 2000.

Jonathan P. Rowe, Scott W. McQuiggan, Jennifer L. Robison, Derrick R. Marcey, and James C. Lester. Storyeval: An empirical evaluation framework for narrative generation. In *Intelligent Narrative Technologies II*, pages 103–110, 2009.

Dennis Schofield. The second person: A point of view? the function of the second-person pronoun in narrative prose fiction, 1998. URL http://members.westnet.com.au/emmas/2p/thesis/0a.htm.

Nanda Slabbers. Narration for virtual storytelling. M.Sc. thesis, University of Twente, Enschede, The Netherlands, March 2006. URL http://essay.utwente.nl/56935/.

Ivo Swartjes. *Whose story is it anyway? : how improv informs agency and authorship of emergent narrative*. PhD thesis, Enschede, May 2010.

Ivo Swartjes and Mariët Theune. A fabula model for emergent narrative. In S. Göbel, R. Malkewitz, and I. Iurgel, editors, *Technologies for Interactive Digital Storytelling and Entertainment*, volume 4326 of *Lecture Notes in Computer Science 4326*, pages 49–60, November 2006.

Ivo Swartjes and Mariët Theune. The virtual storyteller: Story generation by simulation. In *Proceedings of the 20th Belgian-Netherlands Conference on Artificial Intelligence (BNAIC)*, 2008.

Hans ten Brinke. Hide and sneak: Perceptions in the virtual storyteller. M.Sc. thesis, University of Twente, Enschede, The Netherlands, 2014. URL http://wwwhome.ewi.utwente.nl/~theune/VS/Afstudeerverslag-Hans-ten-Brinke.pdf.

Tom Trabasso, T. Secco, and P.W. Van Den Broek. Causal cohesion and story coherence. In *Learning and comprehension of text*. Hillsdale, NJ: Erlbaum, 1984.

Leonoor Van der Beek, Gosse Bouma, Rob Malouf, and Gertjan Van Noord. The Alpino dependency treebank. *Language and Computers*, 45(1):8–22, 2002.

Worldwide Web Consortium. RDF primer, February 2004. URL http://www.w3.org/TR/rdf-primer/.

R. Michael Young and Johanna D. Moore. Dpocl: A principled approach to discourse planning. In *In Proceedings of the 7th International Workshop on Natural Language Generation*, pages 13–20, 1994.

R.M. Young. Story and discourse: A bipartite model of narrative generation in virtual worlds. *Interaction Studies*, 8(2):177–208, 2007.

René Zeeders. Narrator - system description (internal document), 2008.