The automatic generation of narratives

Mariët Theune[1]*, Nanda Slabbers*[1] *and Feikje Hielkema*[2]

[1]University of Twente, Enschede, The Netherlands.
[2] University of Aberdeen, Scotland, UK.*

Abstract

We present the Narrator, a Natural Language Generation component used in a digital storytelling system. The system takes as input a formal representation of a story plot, in the form of a causal network relating the actions of the characters to their motives and their consequences. Based on this input, the Narrator generates a narrative in Dutch, by carrying out tasks such as constructing a Document Plan, performing aggregation and ellipsis and the generation of appropriate referring expressions. We describe how these tasks are performed and illustrate the process with examples, showing how this results in the generation of coherent and well-formed narrative texts.

1 Introduction

Most natural language generation (NLG) systems are aimed at 'serious' applications such as the generation of weather reports, instructions, descriptions of museum artifacts, etc. The automatic generation of narratives, however, is still a largely unexplored subject. A notable exception to this is the work by Charles Callaway on STORYBOOK (Callaway 2000), a full-fledged NLG system for narrative prose generation that can generate many different retellings of the same story (Little Red Riding Hood). The input for STORYBOOK consists of a number of plot arcs selected based on user-specified parameters; the system was never coupled to a digital storytelling system that could generate original plots. Other work addressing the generation of narratives is that by Lönneker (2005), who proposed an architecture for a "narratologically enhanced NLG system" to be used in combination with a story (plot) generator. However, this architecture has not been implemented. One of few systems that have been actually implemented and used as a language generation component in a digital storytelling system is PRINCE, which is used as a front-end to the Proto-Propp plot generation system (Gervás, Díaz-Agudo, Peinado and Hervás 2005). Language generation in this system is based on templates and schema's; a distinguishing feature is its capacity to generate analogies (Hervás, Pereira, Gervás and Cardoso 2006).

In this paper we present another system for narrative generation: the Narrator, the NLG component of the Virtual Storyteller story generation system. We discuss its architecture and give an overview of how the different NLG tasks are carried out. Then we discuss two example stories generated by the Narrator, followed by some concluding remarks and pointers to future work. First, we briefly describe the Virtual Storyteller, the storytelling system of which the Narrator is a part.

*Feikje Hielkema carried out this work while she was at the University of Groningen.

2 The Virtual Storyteller

The Virtual Storyteller[1] is a multi-agent system that automatically creates fairy tales. Story generation in the Virtual Storyteller takes place in three stages, each handled by specialized agents.

The first stage is *plot generation*, which is based on the actions of semi-autonomous character agents in a simulated story world. These agents can reason logically and make plans to achieve their personal goals. In reaction to events and objects, they can experience emotions such as joy and distress, love and hate, and their subsequent actions are influenced by these emotions (Theune, Rensen, op den Akker, Heylen and Nijholt 2004). Note that this is a so-called 'emergent narrative' approach (Aylett 1999) where stories are created by the characters, not based on a pre-authored plot or a story grammar.

During plot generation, a formal representation of what happens in the story world is constructed, called the Fabula (Swartjes and Theune 2006). When all events in the Story World have played out, the Fabula is passed on to the next stage: *narration*. This part of the story creation process is carried out by the Narrator agent, which maps the Fabula to a Dutch text using knowledge about discourse structure and Dutch syntax and morphology. In the rest of this paper, the workings of the Narrator will be discussed in some detail. The third and last stage is *presentation*: an embodied agent representing a human storyteller presents the narrative to the audience using text-to-speech. Our work on the generation of speech with a storytelling speech style is described in Theune, Meijs, Heylen and Ordelman (2006b) and will not be discussed here.

3 The Narrator architecture

The design of the Narrator is based on the pipe-lined NLG architecture described by Reiter and Dale (2000), who distinguish three stages in the NLG process:

1. **Document planning**: determining what is to be said, and creating an abstract document specifying the structure of the information to be presented.
2. **Microplanning**: fleshing out the document specification by the generation of referring expressions, lexicalisation (word choice), and aggregation.
3. **Realisation**: converting the abstract document specification to real text, using knowledge about syntax, morphology, etc. In addition, mark-up may be added for use by external components.

Figure 1 shows the global architecture of the Narrator. It has three modules, corresponding to the three NLG stages described above: a Document Planner, a Microplanner and a Surface Realizer. The Document Planner receives a Fabula as input and turns it into a Document Plan, consisting of plot elements linked by rhetorical relations. The Microplanner converts the Document Plan into a so-called *Rhetorical Dependency Graph* by mapping the plot elements to partially

[1] `http://wwwhome.cs.utwente.nl/~theune/VS/index.html`

lexicalised Dependency Trees. Finally, the Surface Realizer performs syntactic aggregation and the generation of referring expressions, and also takes care of linearization, morphology and punctuation to produce a proper surface form.

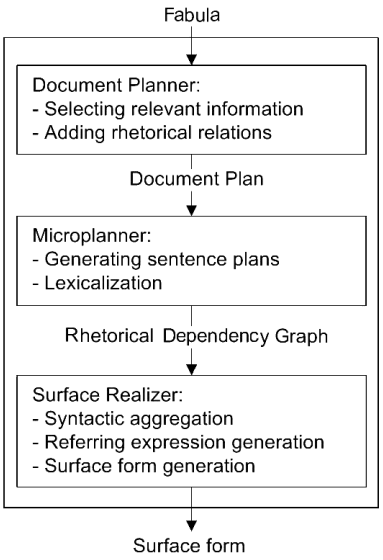


Figure 1: Architecture of the Narrator

The architecture of the Narrator deviates from the 'standard' NLG architecture of Reiter and Dale (2000) in that we situate syntactic aggregation in the Surface Realizer, whereas Reiter and Dale see aggregation as a Microplanning task. Cahill and Reape (1999) investigated the architecture of over 20 NLG systems and found that the location of the aggregation process varied widely across these systems. This divergence is partly caused by the fact that many, quite different processes are gathered under aggregation (Reape and Mellish 1999). However, in the Narrator we only focus on syntactic aggregation, which deals with grammatical processes and therefore in our view should be situated in the Surface Realizer. A consequence of this decision is that the generation of referring expressions is also located in the Surface Realizer: it would not be efficient to generate referring expressions that are at risk of later being deleted during ellipsis (which is part of the aggregation process). More importantly, to generate pronouns, the exact position of their antecedents has to be known.

4 Document Planning

The input for the Document Planning stage of the Narrator is a Fabula (Swartjes and Theune 2006): a causal network representing the story that emerged from the actions of the character agents in the story world. The Fabula does not form a com-

plete network of everything that happened in the course of the story, but captures only those elements that have either a cause or an effect. Our model of Fabula structure is an adapted version of the story comprehension model of Trabasso, Van den Broek and Suh (1989). It has been implemented as an OWL ontology[2] and includes the following plot elements: actions, events, perceptions, goals, outcomes of goals, and characters' 'internal elements' such as emotions and beliefs. The possible relations between these plot elements are motivation, enablement, mental and physical cause relations. Also, each plot element is associated with a time stamp (in terms of discrete, virtual time steps in the story world) from which temporal relations between elements can be derived.

The Document Planner receives a Fabula as input and turns it into a Document Plan by mapping the causal links to appropriate rhetorical relations, removing irrelevant information and adding background information when necessary. We will illustrate this using the (simplified) example Fabula given in Figure 2. This Fabula represents a simple story about a dwarf who is hungry and believes there is an apple in the house. Combined, these two internal elements give rise to the goal to eat the apple. To achieve this goal, the dwarf carries out a simple plan: to take the apple and then eat it. Eating the apple leads consecutively to the perception and the belief that the apple has been eaten, which means a positive outcome for the original goal. Because the Fabula contains several elements that are relevant for plot generation but not for narration, the first step of the Document Planner is to prune away this irrelevant information. A typical example of this is the perception-belief-positive outcome chain following the action of eating the apple in the example Fabula: for the narration it is sufficient to mention only that the action was carried out, leaving it to the reader to infer the rest. Negative outcomes, however, are never pruned as these are generally quite relevant for the story.

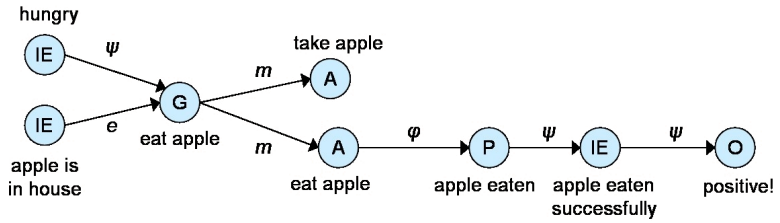


Figure 2: Example Fabula. (IE = internal element, G = goal, A = action, P = perception, O = outcome, ψ = physical cause, e = enablement, m = motivation, ϕ = psychological cause)

The next step is to convert the pruned Fabula to a binary tree and to replace the causal links with appropriate rhetorical relations, inspired by Rhetorical Structure Theory (RST) (Mann and Thompson 1987). The basic set of rhetorical relations used in the Narrator are Cause, Contrast, Temporal and Additive relations, with

[2] `http://www.w3.org/TR/owl-features/`

more specific relations such as Purpose and Elaboration as their subclasses.[3] When mapping the relations in the Fabula to rhetorical relations, consecutive steps of a plan are connected using a Temporal relation; motivation and psychological cause relations are mapped to Volitional Cause relations, and enablement and physical cause relations are mapped to Non-volitional Cause relations. Additive is the most general relation. It is used if two plot elements together cause another plot element, and more in general to connect two plot elements in the Document Plan if no more specific relation holds between them. The automatic assignment of Contrast relations is a subject of ongoing research.

The final step is to extend the Document Plan with information that is relevant for Narration, but which is not specified in the Fabula. Examples are information about the setting (introducing characters and locations) and information on the properties of characters and objects. In Figure 3, which shows the Document Plan corresponding to the Fabula from Figure 2, such added elements are shown in grey: a Setting element introducing the protagonist, connected via an Elaboration relation with an element specifying the protagonist's name. These added plot elements stand in a 'Temporal-once' (*Once upon a time...*) relation to the other elements; this particular relation was added specifically for the fairy tale domain.[4]

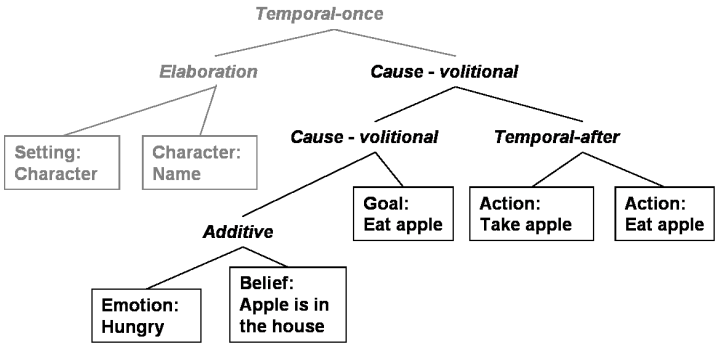


Figure 3: Document Plan based on the example Fabula from Figure 2.

5 Microplanning

The Microplanner maps the plot elements in the Document Plan to partially lexicalised Dependency Trees. We call the result a *Rhetorical Dependency Graph*: a graph (or rather, tree) structure with Dependency Trees expressing simple propositions as leaves, connected by rhetorical relations as nodes. Dependency Trees

[3] Penning and Theune (2007) show that almost all cue words found in fairy tales fit into these classes.

[4] As pointed out by one of our reviewers, it might be more appropriate to classify this relation as Background. However, since it is used only for this one construction, its exact classification is somewhat academic.

an attractive formalism for use in the Narrator, in particular for the purpose of aggregation and ellipsis (see Section 6), because of the independence of word order, and the dependency labels that specify which role a node performs in its parent syntactic category.

In order to convert plot elements to Dependency Trees, sentence templates have been created that specify exactly how the arguments of the plot element should appear in the Dependency Tree. In total, over 30 different templates are currently available to express actions, events, failed actions, settings, states, beliefs and perceptions. Actions and events are expressed using a straightforward active voice construction, with an optional PP argument to express instruments, e.g., *De ridder opende de poort (met een sleutel)* (The knight opened the gate (with a key)). Failed actions are expressed using a complex sentence with *proberen* (to try) as the main verb, e.g., *De ridder probeerde de poort te openen* (The knight tried to open the gate). For internal states, we have standard constructions such as *De prinses was bang* (The princess was scared) and *De kabouter had honger* (The dwarf was hungry). In addition, templates are available for two specific storytelling-style constructions that allow for the expression of high-intensity emotions: sentences such as *Wat was ze gelukkig!* (Oh, how happy she was!) and *Ze was nog nooit zo gelukkig geweest!* (She had never been so happy before!). Such information concerning characters' emotions is usually included as an Elaboration relation in the Document Plan. Another specific storytelling construction is used for the setting: *Er was eens...* (Once upon a time, there was...). Various templates are available for different goal types such as Attain goals, where the agens wants to perform some action or achieve some state (*Hij wilde de appel opeten / gelukkig zijn / de appel hebben*) (He wanted to eat the apple / be happy / have the apple) and Sustain goals, where the agens wants to maintain some existing situation (*Hij wilde blijven eten / gelukkig blijven / de appel houden*) (He wanted to keep eating / remain happy / keep the apple).

Once the sentence templates are selected, the trees are partially lexicalised. References to entities are not lexicalised, as this is part of the generation of referring expressions, which is done at a later stage. All other concepts are mapped to Dutch words by the lexical choice algorithm, which makes use of a discourse history to achieve some variation in wording, taking into account which words have been used recently.[5] The words added to the Dependency Trees are still uninflected, as morphology is taken care of during Surface Realization.

6 Aggregation

To achieve coherent output texts that are more than a sequence of simple sentences, syntactic aggregation is applied to the trees in the Rhetorical Dependency Graph. The aggregation algorithm goes through the graph depth-first, trying to combine the Dependency Trees at the leaf nodes. If aggregation succeeds, the graph is updated with a new, complex Dependency Tree replacing the original relation, and

[5] We use a small lexicon that was constructed specifically for our story domain and contains only a few synonyms; for a more sophisticated approach to lexical choice using WordNet, see Hervás et al. (2006).

the algorithm continues looking for relations to transform.

The syntactic aggregation process consists of three steps. First, based on the rhetorical relation between two Dependency Trees, an appropriate cue word is selected that expresses this relation. Then, depending on the properties of the selected cue word, the two Dependency Trees may be joined together using a specific syntactic construction. Finally, the joined Dependency Trees are checked for repeated elements that can be ellipted. In the remainder of this section we briefly outline these steps; a detailed description of the aggregation process is given in Theune, Hielkema and Hendriks (2006a).

For the purpose of cue word selection, a small taxonomy charting only the most prevalent cue words in Dutch has been constructed, using a variant of the substitutability test described by Knott and Dale (1994). The cue words are divided into four main classes, signaling Cause, Temporal, Contrast and Additive relations. Each of these classes is subdivided into more specific subclasses. A cue word from a subclass can always be replaced by a more general cue word in the same category. We have insufficient space to show the taxonomy here, but the original taxonomy (with 38 cue words) is given in Theune et al. (2006a), and an updated version (with 32 cue words) is presented in Penning and Theune (2007).

The rhetorical relation between two Dependency Trees in the Rhetorical Dependency Graph determines which cue words (if any) can be used to aggregate the trees. If the relation has no specific features licensing the use of a specialized cue word, a more general cue word is chosen. It is not necessarily the most specific applicable cue word that gets selected; discourse history plays a part as well. If a cue word has been recently used, it is less likely to get chosen again. The selected cue word determines the structure of the generated sentence(s). If the cue word is a coordinator, a paratactic structure is created, i.e., a construction where two clauses of equal status are coordinated. A new Dependency Tree is constructed with a root labeled 'CONJ' (conjunction). Its child nodes are a coordinator (the cue word) and two conjuncts (the Dependency Trees to be aggregated). If the selected cue word is a subordinator, a hypotactic structure is created. If the cue word is an adverb, the cue word is added to either the first or the second tree in the relation (depending on the cue word), while the trees remain separate.

In the final step, ellipsis, superfluous nodes or branches are removed from an aggregated Dependency Tree. This only applies to paratactic trees, not to hypotactic ones where one of the combined clauses is subordinated to the other. First the identical nodes (if any) in the aggregated Dependency Tree are marked. We use unique identifiers to distinguish different instances of the same concept, so that ellipsis is only applied to nodes with identical referents. When all identical nodes (if any) have been found and marked, it is determined which operations are suitable, for example Conjunction Reduction, where the subject of the second clause is deleted. This operation is illustrated in Figure 4, expressing the Additive relation in the Document Plan of Figure 3. A corresponding surface string would be something like *De kabouter had honger en dacht dat er een appel in huis was* (The dwarf was hungry and believed there was an apple in the house). The other available forms of ellipsis are Gapping (deleting the main verb of the second clause,

e.g., *De prinses at een appel en de kabouter een peer*) (The princess ate an apple and the dwarf a pear), Right Node Raising (deleting the rightmost string of the first clause, e.g., *De prinses ziet en de prins hoort de kabouter*) (The princess sees and the prince hears the dwarf), Stripping (deleting all constituents but one from the second clause, and replacing them by the word *ook* (too), as in *De prinses houdt van appels en de prins ook*) (The princess loves apples and so does the prince)[6] and Constituent Coordination (combining two non-identical constituents into one and deleting the rest of the second conjunct in its entirety, e.g., *De prins en de prinses houden van appels*) (The prince and the princess love apples).

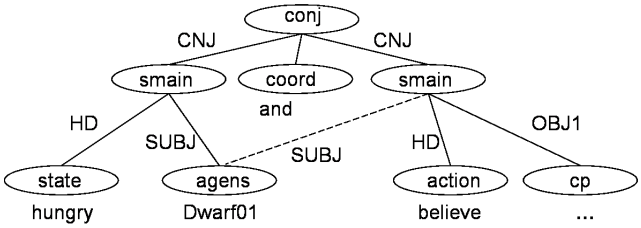


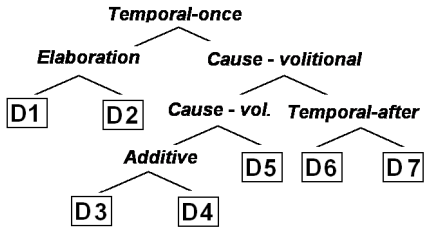
Figure 4: Dependency Tree with Conjunction Reduction.

The aggregation process is recursive in that an aggregated Dependency Tree can potentially be combined with another tree it stands in a relation to. However, to keep the resulting sentences from getting too complex, we added the restriction that at most three simple Dependency Trees can be combined. In cases where this restriction prohibits aggregation it is still possible to express the relation between two Dependency Trees by adding an adverb such as *vervolgens* (then) or *echter* (however) to the second tree. To express the maximum of relations, after the initial traversal and transformation of the Rhetorical Dependency Graph, the algorithm makes another pass through it and expresses some final relations by adding adverbs to non-aggregated sentences.

We now illustrate the aggregation process using our example of the hungry dwarf. Figure 5 shows the Rhetorical Dependency Graph corresponding to the Document Plan from Figure 3, i.e., the same structure but with its leaves replaced by Dependency Trees (here abbreviated using a number). First, the two leftmost Dependency Trees D1 and D2 are combined. They are related by an Elaboration relation, so D2 is attached to D1 as a relative clause. Since the resulting Dependency Tree has a Temporal-once relation node as its parent, the cue phrase *Er was eens* (Once upon a time) is also added to the tree. Next, D3 and D4 are combined, resulting in the tree shown in Figure 4. D5 does not have a sister it can be combined with, so it is skipped in this pass through the tree. (Remember that the algorithm moves from left to right through the Rhetorical Dependency Graph, so D5 cannot be combined with its left sister at this point.) D6 and D7 are related by a temporal relation, so they are combined into a hypotactic structure starting with the

[6]Lit.: The princess loves apples and the prince too.

cue phrase *nadat* (after). In a next pass through the Rhetorical Dependency Graph, the algorithm adds an adjunct to D5: the cue phrase *daarom* (therefore), which expresses the causal relation of the aggregated Dependency Tree from Figure 4 (D3 and D4) to D5. An overview of the result is shown to the right in Figure 5; the full text of the story is given in Section 9.



Er was eens D1 die D2
D3 en D4.
Daarom D5.
Nadat D6, D7.

Once upon a time, D1 who D2.
D3 and D4. Therefore D5.
After D6, D7.

Figure 5: Rhetorical Dependency Graph for the example story.

7 The generation of referring expressions

To decide whether a pronoun can be used to refer to a certain entity, or if it would be better to use a noun, we use an algorithm that combines and extends those of McCoy and Strube (1999) and Henschel, Cheng and Poesio (2000). The algorithm is shown in Figure 6. Its input is the referent r for which a referring expression is to be generated. It returns true if a pronoun should be used and false otherwise. Sometimes, even when a pronoun can be used without ambiguity, it is preferable to use a noun phrase for variation. An analysis of human-written fairy tales led us to a number of conclusions about when a noun phrase is preferred over a pronoun:

- At the beginning of a paragraph.
- If the antecedent has not been mentioned for two sentences.
- If a pronoun has been used a number of times (about four times) and the referring expression is the first one in the sentence.

Also, it is undesirable to use a pronoun when the referring expression should include additional information (e.g., information about the emotional state of a character). This information should be expressed by an adjective or a relative clause, which cannot be combined with a pronoun. If the above conditions do not hold, the algorithm returns true if there is strong parallelism with the previous clause or sentence (Chambers and Smyth 1998) or if the clause in which r appears stands in a Causal relation to the preceding clause (Kehler 2002). Otherwise the algorithm bases its decision on the salience of the referent, which is computed using the salience factors of Lappin and Leass (1994).

If a noun phrase is to be generated, the first step is to decide whether the name of the entity should be used or not (assuming the entity has a name). This decision

Pronominalize(r)

if first reference to r in current paragraph
  or antecedent has not been mentioned for two sentences
  or first reference in sentence and a pronoun has been used 4 times
  or referring expression should contain a relative clause
  or adjective should be added (determined by the Document Planner) **then**
  return false
end if
if r has not been mentioned in current sentence **then**
  if strong parallelism with previous sentence **then**
    return true
  end if
else
  if strong parallelism with first clause
    or r appears in causal relation **then**
    return true
  end if
end if
if r has highest salience value then
  return true
end if
return false

Figure 6: Algorithm used for pronominalization choice.

is made randomly; 25% of the generated references use the name and the other 75% use a description. If the algorithm decides to generate a referring expression containing the entity's name, there are still two possibilities: simply the name (e.g., *Amalia*), or a noun phrase containing the name (*prinses Amalia*) (princess Amalia). The latter construction can only be used when the noun describes a function, such as princess, king or knight. If this is the case the algorithm includes the noun, otherwise it will only generate the name.

If a regular noun phrase is used instead of a name, first a noun has to be selected. To have some variation in the generated texts, for some concepts we have stored some synonyms in the lexicon: a preferred entry (the most commonly used word for that concept) plus one or more additional entries that will only be used occasionally. An example is the concept 'king' with the Dutch word *koning* as the preferred entry and the word *vorst* as an additional entry, which will only be used when the word *koning* has been used a number of times in a row. In addition, for some concepts hypernyms are available that can be used for variation once in a while. For example, *De ridder sloeg de prinses. Het meisje huilde* (The knight hit the princess. The girl cried).

After having selected the noun, three types of adjectives can be added to it:

1. Distinguishing adjectives, which are necessary in order to create an unambiguous referring expression. These are selected using a slightly modified version of the algorithm proposed by Krahmer and Theune (2002). When introducing a new character all known properties of this character are added to the referring expression, because they can be used as distinguishing adjectives later in the story.
2. Adjectives describing a character's internal state.
3. Adjectives that only have a decorative function. These adjectives are only added if the object to be described has no specific properties except its basic type; for example gates and bridges. The Narrator agent maintains a list of adjectives that can be used to 'spice up' the description of such objects, returning cliche expressions such as *een zware poort* (a heavy gate).

The final step of the noun phrase generation algorithm is choosing a determiner and adding this to the noun phrase generated so far. To this end an entity history is maintained. When an entity is mentioned for the first time, an indefinite article is used, and when the entity has been mentioned before, a definite article is used.

The algorithm described above can also create noun phrases that express relations of the referent with other objects, such as *de poort van het kasteel* (the gate of the castle). For the description of the related object, the noun phrase algorithm is applied recursively. In some of these cases, however, the relation can be easily inferred and it would be more appropriate not to mention it explicitly. For example, when the castle has already been mentioned, just saying *de poort* (the gate) is sufficient. Also, in some of these cases a definite article can be used for a first mention, since the entity in question (e.g., the gate) has already been evoked by the mention of the related object (the castle), based on world knowledge ('every castle has a gate'). Such referring expressions are called *bridging descriptions*. To be able to generate this kind of description we have defined a number of inference rules such as $\forall x.\mathit{Castle(x)} \rightarrow \exists y.\mathit{Gate(y)} \wedge \mathit{Has(x,y)}$, which are checked if a referent r is related to another referent r' that has been mentioned earlier. So if r is a gate and r' is a castle that has been mentioned before, the algorithm then checks if there is a rule specifying that an entity of the type of r' usually has an entity of the type of r. If this is the case, then it checks if there is another salient entity that can also have an entity of the same type as r (so it checks if there is another entity that can have a gate – note that this can be another castle, but also an entity of a completely different type). Finally it checks if the entity r' has exactly one r, in which case a definite article can be used; if this is not the case an indefinite article will be used. A similar strategy is used for references to unique entities in the story; for example, in stories it is common to refer to a king as *the king* if there is only one king in the story. Such definite descriptions can be generated by checking if the Story World only contains one entity of this type.

8 Surface form generation

After aggregation and referring expression generation have taken place, the Surface Realiser linearises the Dependency Trees. It traverses the trees depth-first, or-

dering the children of each node by grammar rules that use the syntactic category of the parent node and the dependency labels of the child nodes. For example, the rule: SMAIN → SU + HD + OBJ1 states that if a parent node has syntactic category 'SMAIN' (sentence) and three children with dependency labels 'SU' (subject), 'OBJ1' (direct object) and 'HD' (main verb), then those children should be ordered in the above way. This particular rule would for instance be applied to produce the sentence *De prins zag Amalia* (The prince saw Amalia). Any nouns, adjectives and verbs are inflected at the moment they are linearised. Punctuation is added once linearisation is complete.

This concludes our description of the language generation process in the Narrator; more details can be found in Slabbers (2006).

9 Some example stories

After referring expression generation and surface realization have been applied, our simple example story about the hungry dwarf is finally narrated as follows:

> *Er was eens een kabouter die Plop heette. Hij had honger en dacht dat er een appel in een huis was. Daarom wilde hij de appel eten. Nadat Plop de appel had opgepakt, at hij de appel.*[7]

Note that the Referring Expression algorithm generates the indefinite noun phrase *een huis* (a house) instead of the bridging description *het huis* (the house), which would have been more appropriate if the house in question was Plop's house (which seems a reasonable assumption). However, in this case the Narrator lacked knowledge about the owner of the house and therefore produced a general description. Apart from this error, the output story is well-formed and coherent. But it is also very simple, and therefore we also show a more sophisticated example, generated from a hand-made Document Plan (shown in Figure 7). This input Document Plan contains Contrast relations and paragraph boundaries that cannot currently be generated automatically by the Document Planner, so this example illustrates the output level that could be achieved by the Narrator (in particular, the Microplanning and Surface Realisation components) once these remaining Document Planning problems are resolved.

> *Er was eens een 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.*
>
> *De prinses woonde in een groot kasteel. Op een nacht ging de ridder naar het kasteel. Hij probeerde de zware poort te openen, maar die was op slot.*
>
> *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.*

[7] Once upon a time there was a dwarf who was called Plop. He was hungry and believed there was an apple in a house. Therefore he wanted to eat the apple. After Plop had taken the apple, he ate it.

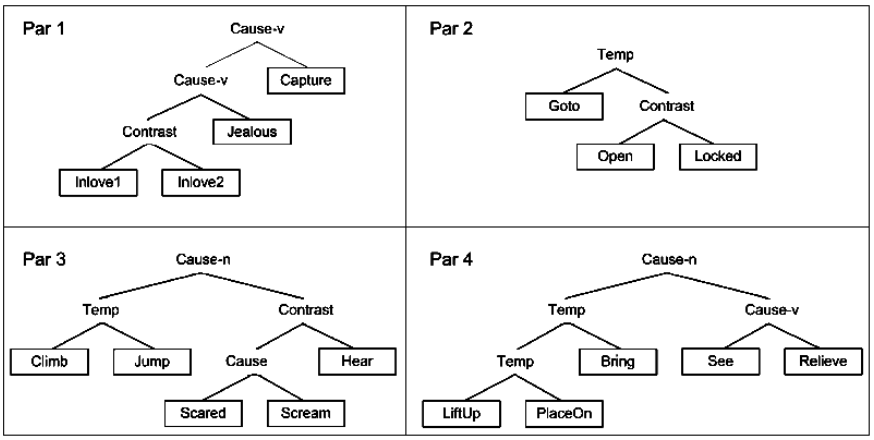


Figure 7: Initial Document Plan for the second example story.

> *De ridder pakte de 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!*[8]

This example story illustrates most of the NLG tasks described above, such as the addition of background information to the Document Plan (at the start of the first and second paragraphs), choice of cue words and aggregation, pronominalization and the expression of 'decorative' properties (*een groot kasteel, een hoge boom*) (a big castle; a high tree) and the use of specific storytelling constructions.

10 Conclusions and future work

In this paper we have presented the Narrator, a natural language generation component designed for use in a digital storytelling system, the Virtual Storyteller. The Narrator has been implemented (in Java), but it has only been tested with handmade input structures, because parts of the Document Planner and of the Virtual Storyteller's plot generation component are still under construction. So far, the only evaluations have been informal comparisons with the output of earlier versions of the Narrator.

[8]Once upon a time there was a beautiful princess who was called Amalia. A knight from a far away country was in love with her, but she was in love with a young prince. The knight was jealous, so he wanted to abduct her. <P> The princess lived in a big castle. One night the knight went to the castle. He tried to open the gate, but it was locked. <P> After the knight had climbed a high tree, he jumped into the princess' bedroom. She was so scared that she screamed loudly, but nobody heard her. <P> The knight grabbed the princess and then he placed her on his horse. After that he took her to an old and narrow bridge. On the other side she saw the prince whom she was in love with. Oh, how relieved princess Amalia was!

The Narrator shows that the pipeline NLG architecture of Reiter and Dale (2000) can very well be used for the generation of narratives. It employs sophisticated algorithms for NLG tasks such as aggregation and the generation of referring expressions, enabling it to generate well-formed and fluent texts. This stands in contrast to the output of most digital storytelling systems, which usually consists of a straightforward mapping of plot elements to fixed expressions.

Unlike the STORYBOOK system (Callaway 2000), the Narrator cannot handle typical properties of narrative prose such as multiple viewpoints or character dialogue, and neither does it employ the type of narratological knowledge as the narrative generation architecture proposed by Lönneker (2005). However, it is capable of generating several linguistic constructions that are typical for fairy tale-like stories, and some narrative generation tasks are currently being investigated. These include the automatic placement of paragraph boundaries, detection of contrast relations and the lexical expression of emotions (taking the intensity of the emotion into account). Also, we would like to extend the Narrator so that it can also generate narratives in English. Since most algorithms and representations used in the Narrator are language independent, we expect that this should be relatively easy to accomplish by replacing the lexicon and the syntactic and morphological rules used for surface form generation.

Our main long-term challenge is to generate texts that are not only grammatical and coherent, but that can also really affect the reader by employing narrative techniques such as the use of subjective perspectives to heighten identification, and foreshadowing to increase suspense. Ablation tests in the style of Callaway (2000) could then be used to evaluate the effect of such techniques.

References

Aylett, R.(1999), Narrative in virtual environments – towards emergent narrative, *Proceedings of the AAAI Fall Symposium on Narrative Intelligence*, pp. 83–86.

Cahill, L. and Reape, M.(1999), Component tasks in applied NLG systems, *Technical Report ITRI-99-05*, Information Technology Research Institute, ITRI, Brighton, UK.

Callaway, C.(2000), *Narrative Prose Generation*, PhD thesis, North Carolina State University, Raleigh, NC.

Chambers, G. and Smyth, R.(1998), Structural parallelism and discourse coherence: A test of Centering Theory, *Journal of Memory and Language* **39**, 593–608.

Gervás, P., Díaz-Agudo, B., Peinado, F. and Hervás, R.(2005), Story plot generation based on CBR, *Knowledge-Based Systems* **18**(4-5), 235–242.

Henschel, R., Cheng, H. and Poesio, M.(2000), Pronominalization revisited, *Proceedings of COLING*, pp. 306–312.

Hervás, R., Pereira, F., Gervás, P. and Cardoso, A.(2006), Cross-domain analogy in automated text generation, *Proceedings of the Third joint workshop on Computational Creativity, ECAI'06*, Trento, Italy.

Kehler, A.(2002), *Coherence, Reference, and the Theory of Grammar*, CSLI Publications.

Knott, A. and Dale, R.(1994), Using linguistic phenomena to motivate a set of coherence relations, *Discourse Processes* **18**(1), 35–62.

Krahmer, E. and Theune, M.(2002), Efficient context-sensitive generation of referring expressions, *in* K. van Deemter and R. Kibble (eds), *Information Sharing: Reference and Presupposition in Language Generation and Interpretation*, CSLI Publications, pp. 223–264.

Lappin, S. and Leass, H.(1994), An algorithm for pronominal anaphora resolution, *Computational Linguistics* **20**(4), 535–561.

Lönneker, B.(2005), Narratological knowledge for natural language generation, *Proceedings of the 10th European Workshop on Natural Language Generation (ENLG-05)*, Aberdeen, Scotland, pp. 91–100.

Mann, W. and Thompson, S.(1987), Rhetorical structure theory: A theory of text organization, *Technical Report ISI/RS-87-190*, ISI: Information Sciences Institute, Los Angeles, USA.

McCoy, K. and Strube, M.(1999), Generating anaphoric expressions: Pronoun or definite description?, *Proceedings of the ACL Workshop on The Relation of Discourse/Dialogue Structure and Reference*, pp. 63–71.

Penning, M. and Theune, M.(2007), Cueing the virtual storyteller: Analysis of cue phrase usage in fairy tales, *Proceedings of the 11th European Workshop on Natural Language Generation (ENLG'07)*.

Reape, M. and Mellish, C.(1999), Just what is aggregation anyway?, *Proceedings of the 7th European Workshop on Natural Language Generation (ENLG'99)*, pp. 20–29.

Reiter, E. and Dale, R.(2000), *Building Natural Language Generation Systems*, Cambridge University Press, Cambridge.

Slabbers, N.(2006), *Narration for virtual storytelling*, Master's thesis, University of Twente.

Swartjes, I. and Theune, M.(2006), A Fabula model for emergent narrative, *Technologies for Interactive Digital Storytelling and Entertainment (TIDSE)*, Lecture Notes in Computer Science 4326, Springer-Verlag, pp. 95–100.

Theune, M., Hielkema, F. and Hendriks, P.(2006a), Performing aggregation and ellipsis using discourse structures, *Research on Language and Computation* **4**(4), 353–375.

Theune, M., Meijs, K., Heylen, D. and Ordelman, R.(2006b), Generating expressive speech for storytelling applications, *IEEE Transactions on Audio, Speech and Language Processing* **14**(4), 1137–1144.

Theune, M., Rensen, S., op den Akker, R., Heylen, D. and Nijholt, A.(2004), Emotional characters for automatic plot creation, *in* S. Göbel and et al. (eds), *Technologies for Interactive Digital Storytelling and Entertainment (TIDSE)*, Lecture Notes in Computer Science 3105, Springer-Verlag, pp. 95–100.

Trabasso, T., Van den Broek, P. and Suh, S. Y.(1989), Logical necessity and transitivity of causal relations in stories, *Discourse Processes* **12**, 1–25.