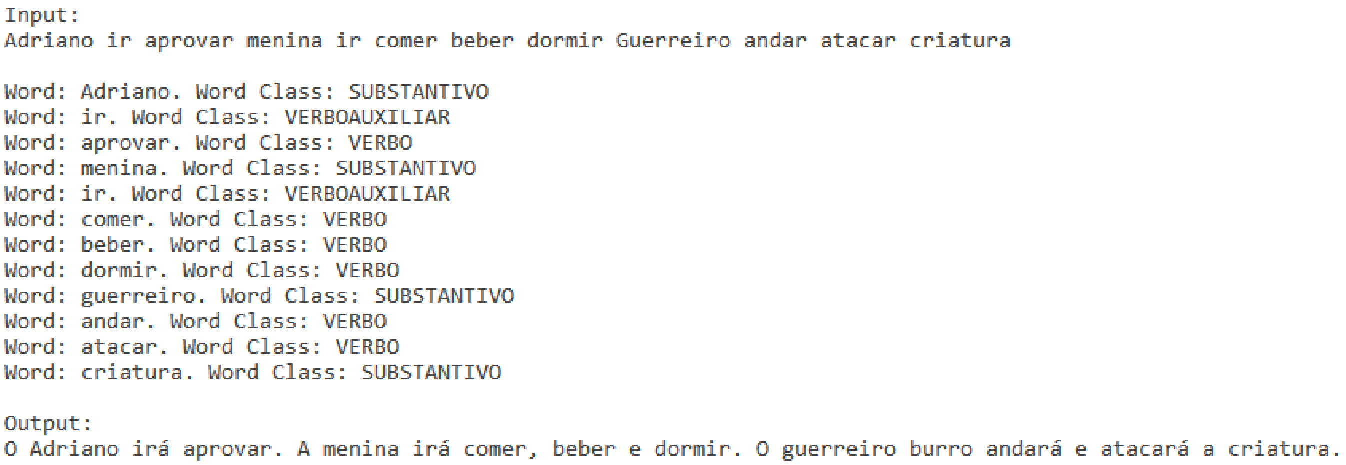
kNarrator: A Model for Authors to Simplify Authoring Process Using Natural Language Processing to Portuguese

Adriano Kerber* Daniel Camozzato Rossana Queiroz Vinícius Cassol

Universidade do Vale do Rio dos Sinos (Unisinos), Escola Politécnica, Brasil



```
Input:
Adriano ir aprovar menina ir comer beber dormir Guerreiro andar atacar criatura

Word: Adriano. Word Class: SUBSTANTIVO
Word: ir. Word Class: VERBOAUXILIAR
Word: aprovar. Word Class: VERBO
Word: menina. Word Class: SUBSTANTIVO
Word: ir. Word Class: VERBOAUXILIAR
Word: comer. Word Class: VERBO
Word: beber. Word Class: VERBO
Word: dormir. Word Class: VERBO
Word: guerreiro. Word Class: SUBSTANTIVO
Word: andar. Word Class: VERBO
Word: atacar. Word Class: VERBO
Word: criatura. Word Class: SUBSTANTIVO

Output:
O Adriano irá aprovar. A menina irá comer, beber e dormir. O guerreiro burro andará e atacará a criatura.
```

Figure 1: kNarrator's result using multiple sentences in a single pseudo-text.

Abstract

In this paper, we propose a model to help writers to produce narratives or text fragments using only few words as input. The proposed model uses pseudo-text as input and returns full fluid text as output. The facts described in the pseudo-text can be transcribed with a different level of detail set by the user, from a simple sentence to an expanded description, adding details according to a user-defined semantic dictionary. This allows authors to visualize ideas and concepts of narratives. Our model can also be used in games as a tool to generate narratives and descriptions in natural language text. In order to evaluate our approach, we performed a comparative study with some authoring models and results are further discussed.

Keywords: Natural Language Processing, Natural Language Generation, Authoring Model, Portuguese, Storytelling.

1 Introduction

There has been a growth of research focusing on authoring models in recent years. Most research done has focused on improving the way authors create their own narratives, stating that individuals or teams of writers need to tirelessly create huge amounts of content by hand, which is impractical for full length narratives and game titles. Different techniques have been proposed to help authors in the creation process, and we proposed a taxonomy which divided the techniques in two types:

- Not plot-based: The author define the entire creation with one type of rules.
- Plot-based: The author has different levels of rules to describe each character or element that can be presented.

In the not plot-based techniques, the author creates text pieces and simple rules that may rearrange the text pieces in the final composition. Then the rules created by the author are responsible directly for the final composition and the task of plot control is inside the author's mind.

This is the case of the Tracery model [2]. In a first step, the user creates sets of words and rules. In a second step, these rules are used to choose words from the sets to randomize specific words in an annotated sentence. Another not plot-based model is Expressionist [14], which also uses rules and sets of words for the annotated sentences, with the difference that the order in which the words are retrieved from the set is probabilistic instead of random.

Another interesting not plot-based model described in a platform study from Friedhoff [3] it the Twine. Twine [6] is a system inspired by interactive fiction which allows the author to create branched stories in a visual way, allowing the author to see the connections among the branches.

Thus the previous models are not plot-based, they do not treat the narrative structure as part of the model. They allow the author to deal with the plot organization by himself.

On the other hand, there are the plot-based techniques, where the author still creates text pieces and rules, but the text pieces are organized by complex rules with different types and levels of usage. Then each type of rule on plot-based is directly related to plot structures definition such characters or situations.

This is the case of the Wide Ruled 2 model [15], in which the organization of the stories is completely controlled by the system, enabling the author to focus on writing characters, worlds and story plots in its data structures.

The main difference between the previous models and Wide Ruled 2 is that the previous concern themselves with giving the

*e-mail:kerberpro@gmail.com

author almost full control over the narrative with no focus on the plot, while in Wide Ruled 2 the main point is to enable the author to simply set preferences and the story is generated by the system itself.

In all the reviewed models, the author creates the text fragments and templates, which the system uses to generate the final text in natural language. Then all the knowledge of natural language is expressed by the author. Thus, we could identify a lack of a model which uses a natural language processing module (NLP) within the textual creation pipeline, to allow the creation of the final text automatically.

We propose a model that uses a simple approach to text creation, such as in the Tracery model [2], in addition our model inhibits the necessity of learning a specific syntax which is present in all the previous models. The model will allow the author to have full control over the narrative and at the same time, it will not be necessary to create a full text such as in the previously presented models. The focus of our model, named kNarrator, is to put the task of natural language processing directly within the model's pipeline. This allows the author to create text, without needing to focus on the details related to generating the final text.

For this purpose, kNarrator does receive pseudo-text as input, as well as details about words and context for the input to generate a full fluid text with as much detail as the author wants.

It is important to notice that our work is currently focused on the Portuguese language, but on the other hand, our model can be easily adapted to other languages.

2 Related Work

In this section we present the most relevant researches found until this paper was written. The Twine model [6] is an approach to creating interactive fictions. In this model, a graph is used to give structure to the narrative. Each node in this graph contains a fragment of text, and can lead to other nodes. The sequence of events is defined by the author in a visual editor. First, the author manually divides the text into text boxes. Then, the text boxes can be linked, such that each text box is a piece of the narrative that can be linked to different text boxes. This allows the author to create interactive narratives, with text fragments that lead to different text fragments.

Thus, Twine enables the creation of user-defined (fixed) text variations. Twine also handles user-defined rules such as programming code and variables, which can be used to define specific behaviors. For example, “enable this node if the character has 3 gold stones”.

The Twine model differs from our model by the use of nodes and links, and user-defined rules. In our case we use pseudo-text that after processed becomes a linear full fluid text, as described in Section 3.

The Tracery model [2] has a different approach from Twine's. Tracery uses two types of information. The first type holds different sets of words, each with the same general purpose. Examples of such sets would be names, nationalities, genders, greetings, etc. The second type are text fragments. These text fragments can contain tags which indicate places where a tag will be replaced with words from a specific set.

Thus, the final text is formed from a simple rule, which replaces tags with a random word from a specific set. The Tracery model differs from our model with the concept of user-defined rules and the use of templates to generate the final text.

The Expressionist model [14] has a visual editor and a similar approach to Tracery [2], in that both use sets of words and rules. This model also uses text fragments containing tags which can be replaced by a word from a set of words. The difference is that Expressionist allows the user to assign a value to each word in a set. These values are then used as weights to select words with a probabilistic order. This is different from Tracery, which selects elements randomly.

In addition, Expressionist provides an editing tool to facilitate the workflow of the author. The tool is organized in panes, and each pane is used to organize a step of the production flow. There are four panes in the tool: *in*, *todo*, *write* and *out*. The in pane is used to populate a list of deep representations, which are structured representations of the semantic content of a sentence. The todo pane receives the list of deep representations from the in pane. The write pane is used to specify production rules, which are sets of words with weights defined for each word. Finally, the out pane is used to export the resulting database, defined by the author, into a structured format, enabling its use with other applications.

The Expressionist model differs from ours in relation to the definition of the rules and templated text, which are characteristics that we avoided to have in our model to achieve the goal of NLP. Expressionist has an interesting interface to treat its deep representations that could be used as a reference for a future visual interface for editing the semantic dictionary of our model.

The Wide Ruled 2 (Wide Ruled) model [15] is a story authoring tool that attempts to reduce the technical expertise required from the user and creates a bridge between algorithms and art by providing a non-technical, writer-oriented authoring interface to a text-based interactive story generator.

The Wide Ruled model is templated, as were all the previous models, but with a defined structured interface that allows the author to create the elements, plot points and goals to define the generation of the narrative. The visual editor of Wide Ruled is divided in four panes:

1. Characters - this pane is filled by the author with a list of characters that can have attributes and relationships with other characters;

2. Environments - where a list of scenarios that can have attributes and relationships with other scenarios;

3. Plot Point Types - a pane where plot points can be defined with their attributes;

4. Goal and Plot Fragments - is the pane where the story is structured and prepared by the author. This pane has a tree like structure to organize and prepare the narrative for the final generation of Wide Ruled. The elements of this final pane are: *author goal*, that is an initial point (root of the tree structure) for the story that is always executed; *plot fragment*, the element that can be selected by Wide Ruled and that takes place under an author goal; and *story actions*, the nodes that are sequentially-executed under a plot fragment.

In a first step the author fills the lists of characters, environments and plot point types. In a second step the author uses the Goals and Plot Fragments pane to organize and use the elements created. In this step the author creates author goal and plot fragments with its story actions.

Wide Ruled is a versatile model to authoring process but difficult for beginners since it demands from the author a knowledge of all the control structures of the model.

Wide Ruled differs from our model in that we do not aim to define a plot-based structure. Despite the focus difference, Wide Ruled has an interesting concept of plot-based generation of the text that we used in our model, but with a different approach.

Other related works are: the model presented in [13], which enables querying Probabilistic Context-Free Grammars (PCFGs) using an algorithm to construct a Bayesian network from PCFGs to allow generalized queries; the model presented in [7], that creates a method called the Scheherazade system which generates a simple story using plot graphs learned through crowdsource from stories generated by human authors; and a model called Curveship [9], which is an approach for natural language processing in interactive fictions. This model has a variety of templated sentences that are inserted in a random order in the interactive fictions.

These models can be exploited by our model for the matching process (see Section 3.1) to identify or classify the words from the input.

2.1 Simple taxonomy of Natural Language Processing algorithms

In Natural Language Processing (NLP) the algorithms can be divided in:

- Normalization algorithms: Lemmatization (From [11] or [5]) and stemming [12] are algorithms of normalization. They identify a canonical representative for a set of related word forms. Then it means that the algorithms group words and link them as morphed forms of a base word.

- Word-category disambiguation: Also known as Part-Of-Speech tagging (POS tagging or POST), is a technique that usually uses a dictionary (or *corpus*). The POS tagging analyzes a text to identify the possible word classes of each word or group of words, then based on its positioning on the text define the correct word class for the word in the text. A study of tagging techniques can be read in Part-of-speech tagging [17].

The normalization techniques differ in their approaches. Stemming usually refers to a crude heuristic process that removes the suffixes and affixes of a word to find its stem. On the other hand, the Lemmatization technique usually refers to using a vocabulary and morphological analysis of words, normally aiming to remove inflectional suffixes and returns the base or dictionary form of a word, which is known as the lemma. The stemming techniques are faster but less assertive while the lemmatization techniques, using its vocabulary rules and dictionary, tend to be less efficient but accurate.

The word-category disambiguation techniques can be mainly rule-based or stochastic. A rule-based technique such as "A Simple Rule-Based Part Of Speech Tagger" [1] has its own tagger that classifies the words initially by their most common classes, then starts the process of comparison of the result tags with a percentage of assertion and error from the corpus used and resetting new word classes to each word. The technique goal was to learn from the result set to improve the next results becoming faster then a stochastic method.

Another rule-base technique is "Dependency parsing with compression rules" [4] which combines two POS taggers with compression rules to create a reliable and fast disambiguation model.

3 The kNarrator Model

Our purpose is to create an authoring tool that uses a pseudo-text to generate a full fluid text using natural language processing, and that is able to not only generate this fluid text but also insert new content to expand the output text at runtime. We aim to create a versatile model that can facilitate the creation process for the author by removing the need to learn a complex structure of rules. The kNarrator is currently implemented using the C# language from .NET framework [8], this language was chosen to enable a future integration of the

kNarrator model to game engines as Unity3D [16], which supports the C# language. In addition we use a SQLite database [10] to store the words' data.

The main input of our model is a pseudo-text. The pseudo-text is a term we used to describe an incomplete sentence or text. For example: "Maria ir casa", is a pseudo-text that can be translated to a natural language sentence as "A Maria foi para casa.".

Then a pseudo-text can be primarily described as lemmatized, which means that all the words presented on it are in its lemma form. Although it is important to notice that to facilitate the use for authors, our model allows that the pseudo-text uses inflected forms beyond the lemmatized forms, which inhibits the necessity for the author to learn a specific syntax to use the kNarrator model. our model receives a pseudo-text as input and converts that to a full fluid text in natural language as output (see Figure 2). current goal for kNarrator model is to construct the basic tool, basic pipeline, capable of classifying pseudo-text in Portuguese language and generating the output text in natural language. Then we propose our own implementation of a rule-based model.

Our model is divided in three main modules: the Classifier, the Expander and the Organizer.

3.1 Classifier Module

The Classifier module is where the input pseudo-text provided by the author is analyzed and classified.

Then this module uses a POS tagger technique. Each word on Classifier module is assigned a Word Class and stored in a structure called Token. To classify each word, this module uses a dictionary to search for words matching the input. Thus, each word from the input text is stored alongside its Word Class in a Token.

A Word Class is a representation of a word class in Portuguese and is stored in the Token with its name and an array of all the inflections the word class in question allows.

For a better understanding, the Portuguese language word classes with their respective inflections are: Verb - with inflections in mode, time, number, person and voice; Noun - with inflections in gender, number and grade; Article - with inflections in gender and number; Adjective - with inflections in gender, number and grade; Adverb - with inflection in grade (just in a few cases); Pronoun - with inflections in gender, number, person and case; Numeral - with inflections in gender, number and grade (just in a few cases); Preposition - with no inflections; Conjunction - with no inflections; Interjection - with no inflections.

Based on these word classes we created a base structure with unique identifiers to represent these word classes called Word Class Identifiers. An example of a pseudo-text input could be "Urso atacar homem floresta denso". In this example all the words are in a base form, and our model would classify the words and generate the following Tokens: "Urso" as a noun, "atacar" as a verb, "homem" as a noun, "floresta" as a noun, "denso" as an adjective.

In this case the classification of the words could use a simple dictionary with templated words, to match and classify the words. However, this kind of pseudo-text, written using only the base form of words, is not practical to write. Thus, as explained previously in this section we propose a different approach for our dictionary, such that our model can not only accept pseudo-text, but also regular text, avoiding the need for the user to learn the correct way to write the pseudo-text input.

3.1.1 kDictionary

Our dictionary called kDictionary has pre-classified words stored in a database. Then our dictionary uses a SQLite database [10] to store the words and its inflections.

Our dictionary uses the concept of normalization algorithms, such as lemmatization algorithms, by having the words stored as lemmas and all the respective inflections connected to them.

Thus in our dictionary, the base word, which is a lemma, is connected with the respective inflections, which are saved as inflected words with all their respective classification (Such as gender, number, person, case, *degree*, mode, time and voice) related to identify what the inflection represents for a respective word class.

We divide the tables of the SQLite database in one main table that has all the words and its inflections, and the secondary tables that represent each word class. The main table has the words with

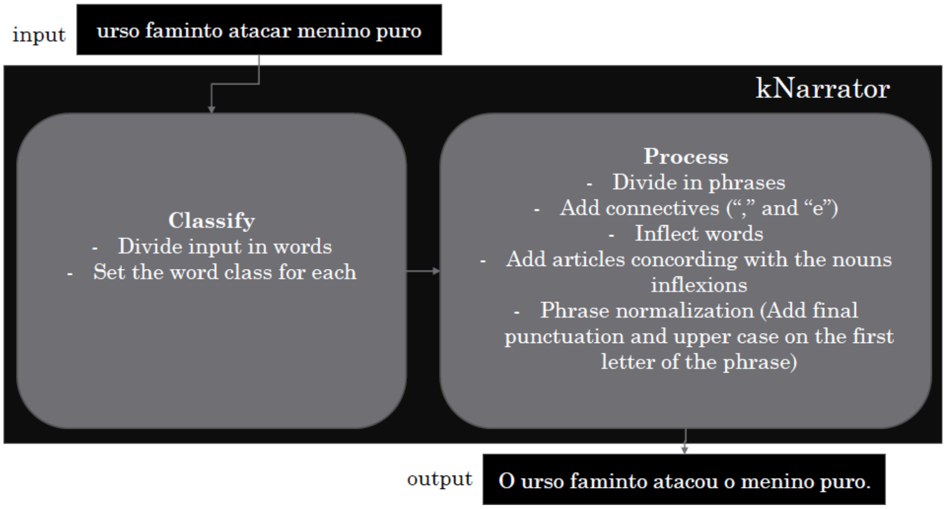

input
urso faminto atacar menino puro
kNarrator
Classify
- Divide input in words
- Set the word class for each
Process
- Divide in phrases
- Add connectives ("," and "e")
- Inflect words
- Add articles concording with the nouns inflexions
- Phrase normalization (Add final punctuation and upper case on the first letter of the phrase)
output
O urso faminto atacou o menino puro.

Figure 2: An overview of the model's base flow, where the input passes on a classification process and a structuring process to generate the output in full fluid text.

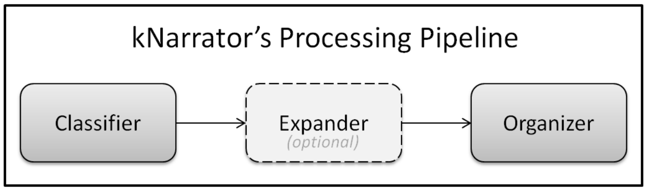


Figure 3: Overall architecture of the kNarrator, showing the regular and expanded flow (dashed line means that the Expander module is optional).

all their possible word classes, then it is divided in two types of words:

- Lemma - a word that represents a lemma and has all its possible word classes listed.
- Inflected - a word that represents an inflection, which is an inflected lemma and has only its word class and the detailed information about what it represents from the lemma.

For example, the lemma "viver" would have an inflected form "viveu". In the dictionary structure this lemma "viver", in the main table, would have all its word classes connected to it, and the inflection "viveu" would have attached to it its classification such as inflection in person (E.g.: first person), number (E.g.: singular), mode (E.g.: indicative), time (E.g.: past) and voice (E.g.: active). Thus our dictionary stores each lemma with all its inflected words pre-classified. Then in the secondary tables we have all the inflected words information connected to the specific lemma.

3.1.2 The POS tagging algorithm

Our POS tagging algorithm uses a preference classification (also called probabilistic), that based on the word position on the pseudo-text the word is more likely to be one word class than other.

The POS tagging steps to classify the input are:

- Step 1: Process the pseudo-text removing punctuation and obtaining all the words
- Step 2: Match all the input words with the words from the dictionary, retrieving all the possible word classes for each word
- Step 3: Find a possible auxiliary verb or verb and set as verb
- Step 4: Classify verb adjacencies searching through all the possible word classes of the current word. The preference order of classes are:
  - Noun
  - Pronoun
  - Article
  - Conjunction

The final result of this step is a list of Tokens. It is important to notice that we ignore the punctuation from the pseudo-text input to show that pseudo-text if well structured can be correctly processed by a rule-based model.

Any unidentified words are passed to another module, called ErrorManager (see Section 3.4).

3.2 Expander Module

The Expander module is responsible for the insertion of new words in the list of Tokens, to expand the text with descriptions for selected words.

The input for this module is the list of Tokens generated by the Classifier module, and the output is a list of Tokens augmented with new words.

In the first step this module analyses the list of Tokens searching through the lemma's word classes for a specific word class from the semantic dictionary (see Section 3.2.1).

Then, in the second step, the Tokens with the corresponding class will be selected to receive a new word.

Thus, in the third step, each one of the selected tokens receive a word from the set of words registered for them.

An example of result, from this module, for the input "Urso atacar homem floresta", is the expansion to "Urso grande forte atacar homem floresta denso silencioso". Where the words "Urso" and "floresta" received an addition of words. It is important to notice that the use of this module (Expander Module) is optional by the author. Then our model can be considered with two different pipelines for generating the full fluid text as an output: 1) Regular pipeline - which generates full fluid text with no textual expansion (see Figure 3). In this pipeline the Expander Module is not used, then the text passes through Classifier Module and go directly to Organizer

Module; 2) Expanded pipeline - which generates full fluid text with textual expansion. In this pipeline the Expander Module is used, then the text passes through Classifier Module, Expander Module and Organizer module respectively.

3.2.1 Semantic Dictionary

The semantic dictionary stores words that represent elements from the narrative world. These elements are defined as new word classes for our kDictionary, the classes are Character ("Personagem") which represents an actor that can perform actions and Place ("Lugar") that is an environment where actions happen. Each lemma registered on the semantic dictionary has a set of words.

The lemmas are represented by nouns while the words on the word sets are adjectives and each token from a word set is obtained via randomic order.

It is important to notice that the semantic dictionary stores the words and concepts from a specific narrative universe. Thus each story needs a dictionary which is adequate for its context.

3.3 Organizer Module

The Organizer module processes the list of Tokens to generate the final text in natural language. The module is responsible for creating meaning for the text, as well as using phrasal rules to insert new tokens and punctuation, removing unnecessary words and reordering words that do not make sense in the current order.

This module can be controlled by the author defining four parameters of inflection control, called Output Parameters, they are:

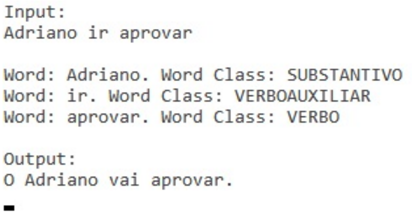
- Person - assuming the values: first, second and third
- Number - assuming the values: singular and plural
- Gender - assuming the values: male and female
- Time - assuming the values: present, past and future (The specific values are: "presente do indicativo", "pretérito perfeito do indicativo" and "futuro do presente do indicativo")

It is important to notice that the previous output parameters do not need to be defined by the author, since a concordance step is done to ensure that all the words are properly inflected. Also, there are default values for the parameters, that are used in the concordance step if possible. The default values for the parameters are: Person as third, Number as singular, Gender as male and Time as present.

This module is divided mainly in five steps for processing the text from the list of Tokens:

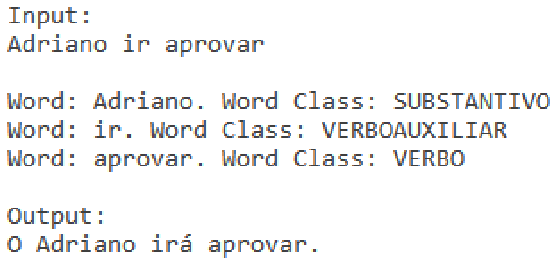
1. Create Sentences - this step divides the tokens in sentences and adds final punctuation. To find the possible sentence the algorithm follows these steps:
   - Beginning of the sentence - search for the first noun
   - Middle of the sentence - search for a verb or auxiliary verb
   - End of the sentence - search for a next verb or auxiliary verb, if a token comparison is found, it goes back from the current verb back to the current verb that represents the middle of the sentence, until it finds a token that breaks a chain of nouns or pronouns or until it reaches the middle sentence verb.
2. Concordance - this step inflects the tokens so that they concord among themselves. In this step the adjacent words are analyzed to keep concordance.
3. Connectives - this step adds connectives to the sentences, such as comma and "e" (and). The connectives are added among repeated word classes. The comma connective is added only if there is no two repeated word classes or more after the insertion point.
4. Articles - this step inserts articles before nouns.
5. Finishing - this step adds capital letter to the beginning of phrases.

The input "Urso atacar homem floresta denso Urso matar homem Urso ir dormir" (with the control parameters: Person = third, Number = singular, Gender = male and Time = past) after processed by this module will result in "O Urso atacou e matou o homem. O Urso matou o homem. O urso foi dormir.".



```
Input:
Adriano ir aprovar

Word: Adriano. Word Class: SUBSTANTIVO
Word: ir. Word Class: VERBOAUXILIAR
Word: aprovar. Word Class: VERBO

Output:
O Adriano vai aprovar.
```

Figure 4: Screenshot from kNarrator's console log with output parameter Time set to present.

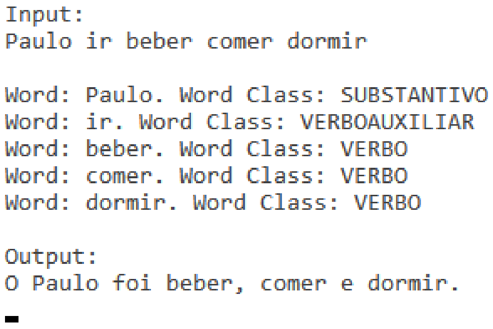


```
Input:
Adriano ir aprovar

Word: Adriano. Word Class: SUBSTANTIVO
Word: ir. Word Class: VERBOAUXILIAR
Word: aprovar. Word Class: VERBO

Output:
O Adriano irá aprovar.
```

Figure 5: Screenshot from kNarrator's console log with output parameter Time set to future.

3.4 ErrorManager Module

The last module in our model is responsible for treating errors. It receives a list of words which could not be recognized by the Classifier module in the input text. Each unidentified word is in a structure that contains the word, the number of the line from the input



```
Input:
Paulo ir beber comer dormir

Word: Paulo. Word Class: SUBSTANTIVO
Word: ir. Word Class: VERBOAUXILIAR
Word: beber. Word Class: VERBO
Word: comer. Word Class: VERBO
Word: dormir. Word Class: VERBO

Output:
O Paulo foi beber, comer e dormir.
```

Figure 6: Screenshot from kNarrator's console log when a sentence with multiple verbs is presented.

text, if divided in lines and the counter that identifies the counting of the word at the line. All this data is passed to the author to facilitate the identification of the error, so the author can either correct or add a new word to the dictionary.

4 Results

Our current goal was to show the capabilities of our model, then we used a console log for the testing purpose as seen in figures 4, 5 and 6. Then the results obtained from our model show that the pseudo-text purposed as input can be used to generate full fluid texts.

It is important to notice that the kNarrator model can be considered multi-language, since the model can be adapted to support other languages by changing or adding new rules. Then we present the results from our model divided in two sets: 1) Regular pipeline (see Figure 3), where the pseudo-text is converted to full fluid text only; 2) Expanded pipeline (see Figure 3), where the pseudo-text is expanded and then converted to full fluid text.

4.1 Regular pipeline results

The results from this pipeline are purely the conversion of our model of pseudo-text to a full fluid text without textual expansion. It is important to notice that all the output parameters with the value "Not set" are going to be ignored on the concordance step described on Organizer Module Section.

The input "homem abraçar menino" with the output parameters set as [Person: Third; Number: Singular; Gender: Not set; Time: Past] generates the output "O homem abraçou o menino“. Other example is the input "homem abraçar menina feio" processed with the parameters [Person: Third; Number: Plural; Gender: Not set; Time: Past] generates the output "Os homens abraçaram as meninas feias".

In this example all the nouns ("homem" and "menina") are inflected to the plural and the adjective "feio" is converted to the plural to concord with the noun "meninas" that preceded, the adjective was inflected also in gender to concord with the gender from "meninas", then the gender was inflected even not being defined by the author, to concord with the previous word.

Another result using the same input "homem abraçar menina feio", but with the output parameters modified to [Person: Third; Number: Plural; Gender: Female; Time: Future] we have the output "As mulheres abraçarão as meninas feias.". In this example the input word "homem" inflected in gender and number became "mulheres", since in the dictionary of kNarrator model we have male and female forms for each noun. The rest of the sentence was inflected as the last example.

4.2 Expanded pipeline results

The results from this pipeline are the results of the additional use of the Expander module to insert in a randomized order new words for the final fluid text. For the examples we used the list of semantic words and its respective word sets below:

- Word: "homem" with the set of words:
  - "sábio"
  - "velho"
  - "cansado"
  - "careca"
  - "cabeludo"
- Word: "guerreiro" with the set of words:
  - "burro"
  - "mau"
  - "bom"
  - "bonito"
  - "feio"

With inflections as [Person: Third; Number: Plural; Gender: Female; Time: Present] set for the input "bode atacou morder abraçarão homem guerreiro" we obtain the result "As cabras atacam, mordem e abraçam as mulheres velhas e as guerreiras bonitas.". In the pseudo-text input we have the verbs either inflected or in the infinitive form and they ("atacou", "morder" and "abraçarão") are recognized and inflected to concord with the output parameters person, number and time. We used the verbs inflected to show the possibility of using words already inflected in the pseudo-text, to show that the author do not need to learn a specific syntax of the pseudo-text input we proposed in the Section 3.

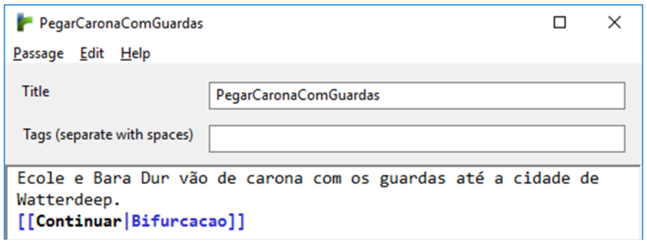
Beyond that, the words "homem" and "guerreiro" (Both nouns) received each randomically an adjective from its own word set. In the current result we had the adjective "velhas" ("velho") to the semantic noun "mulheres" ("homem") and we had the adjective "bonitas" ("bonito") to the semantic noun "guerreiras" ("guerreiro"). Then as the processing randomly selects a word from the word set, if the same input with the same parameters is processed again we could have a different result as "As cabras atacam, mordem e abraçam as mulheres sábias e as guerreiras más.". In this example the selected adjectives were "sábio" and "mau" respectively.

In Figure 1 is shown a result of multiple sentences inserted in a single pseudo-text with only one word registered as a lemma for the semantic dictionary. The word registered is "Guerreiro", that was registered as a Character, which received the adjective "burro" from the semantic dictionary.

4.3 Comparisons

In this section we will make the comparisons from our results and the results from all the other presented authoring models. It is important to notice that since all the compared models use templated texts, they all have repetition patterns on the final text and few re-use of the text fragments, being inferior of our approach that uses natural language processing to create texts word by word, avoiding templated sentences.

4.3.1 Twine



Title: PegarCaronaComGuardas

Tags (separate with spaces):

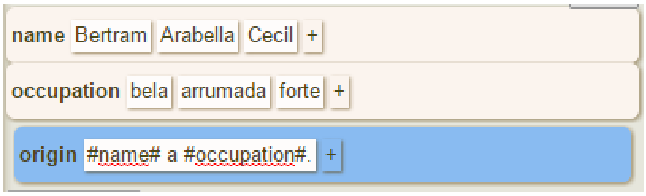
```
Ecole e Bara Dur vão de carona com os guardas até a cidade de
Watterdeep.
[[Continuar|Bifurcacao]]
```

Figure 7: Screenshot from Twine model visual editor.

The Twine model with a structure of nodes as seen in Figure 7 is simple for the author to write the narrative as full text on each node and with a simple syntax, creating connections among other nodes of full text. On the other hand, when the author wants to create and re-use the text fragments created on each node, adapting the text for each case or interaction, the syntax becomes complex since knowledge of programming languages is required by the author. Then, since our model is kept with a simple syntax, the author can create more variations using no specific syntax. Also, another important feature that is not supported by Twine is the capability of inflecting the sentences, since the model does not use a dictionary.

Thus as seen in this comparison, the kNarrator facilitates the text creation and manages the text itself with no requirement of control by part of the author. This features enable the author be free to think more about the content than on the structure itself.

4.3.2 Tracery and Expressionist



name Bertram Arabella Cecil +

occupation bela arrumada forte +

origin #name# a #occupation#. +

Figure 8: Screenshot from the Tracery model

The Tracery model and the Expressionist model, are equally based on simple rules that enable the author to create text, either with random (in Tracery) or probabilistic (in Expressionist) selection of words.

We had access only to Tracery model when making the comparisons with our model, the following explanation is focused on the Tracery model, but since the Expressionist model only varies in matters of selecting the word, both models can be considered the same in the following comparisons.

The rules from these models can be simply the rule “#name# a #occupation#” where the words between hash symbol are variables that identify word sets (see Figure 8). The word set “name” with the words “Arabella”, “Georgia” and “Patricia” and the “occupation” with the words “trabalhadora”, “corredora” can generate the results “Patricia a trabalhadora” or “Arabella a corredora”.

As seen in the comparison with Twine model, the results are simple and demand a time consuming work of textual structuring to generate similar results to the results from our model. It is important to notice that the main difference is that our model creates all the text structuring by itself, which enables the author to care exclusively about the context instead of working on an exhaustive textual structuring process.

4.3.3 Wide Ruled 2

While in the previous models presented in this section creating simple text with simple rules was possible, and to create more complex text a complex syntax was needed, in the Wide Ruled 2 model a complex syntax is mandatory in all the text structuring, since the whole model uses structures that require programming skills and a sequential logic to organize the final text.

In figure 9 a Plot Fragment from Wide Ruled 2 is being shown and it is important to notice that the columns of attributes demand an understanding of programming logic to define the preconditions and story actions properly.

Thus, the Wide Ruled 2 model is more complex than kNarrator because requires programming skills from the author. It is important to notice that this model also shares the same problems from all the previous models, since do not control text structuring, giving this exhaustive task to the author.

The presented facts and comparisons in this Section show that kNarrator is superior to all presented models when concerning to freeing the author from the exhaustive task of text structuring, also giving control for textual variations through word inflections and semantic word sets. Thus with the natural language processing of our model we also surpass the other models in relation to re-use of text fragments for creating more reliable sentences.

5 Final Remarks

In this paper we presented kNarrator, our authoring model that differs from all approaches presented (see Section 2), since all the previous authoring models demand that the author creates all the text fragments and at some level use or learn a specific syntax. In the previous models the author also was required to use some logic to be able to generate consistent results. Thus since our model does not demand the author to learn a specific syntax, accepting variations of the simple syntax for our pseudo-text input, the author can reach more results with less input and with no concern about the text structuring.

Also as our model generates each sentence using idiomatic rules, the capability of text variation occurs at word level, which differs from all the previous models that have variations at sentence level. Thus our model can be considered efficient and less identifiable in terms of pattern repetition when evaluated by the point of view of the reader (The person that will read the final text).

It is important to notice that the kNarrator model was thought to be used along with real time applications, such as games or other interactive systems, in order to enable procedural storytelling in real time.

Another interesting conclusion from the current study was that the kNarrator model could be used together with models as Twine [6] to improve the author experience, approaching a natural language processing capability to a text structuring for branched stories.

5.1 Future Work

As future work the Expander module will be improved together with all the concept of the semantic dictionary integrated, allowing more elaborate descriptions for the words.

We also intend to create a narrative manager algorithm for the Expander module that will manage the text creation allowing narrative creation in a level of characters, scenarios and events.

In Organizer module some difficulties to divide the phrases have been found due to our approach of ignoring punctuation. Then we expect to define with more specialized rules the pseudo-text classification creating a more fluid and variable final text.

Also, our current structure of idiomatic rules is set to the Portuguese language, but we purpose the creation of a symbolic rule system to enable the easy creation of independent idiomatic rules that would enable multi language capabilities. With the symbolic rule system the kNarrator model could load a rule set for a specific context, which could create more flexibility for the creation process also.

Thus, as commented before, a narrative creation via Expander module could be interesting, but with the symbolic rule system, the narrative creation could be implemented and detailed by the author itself if needed. This feature would enable a infinity of possibilities for the textual creation with NLP.

References

[1] E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the workshop on Speech and Natural Language*, pages 112–116. Association for Computational Linguistics, 1992.

[2] K. Compton, B. Filstrup, M. Mateas, et al. Tracery: Approachable story grammar authoring for casual users. In *Seventh Intelligent Narrative Technologies Workshop*, 2014.

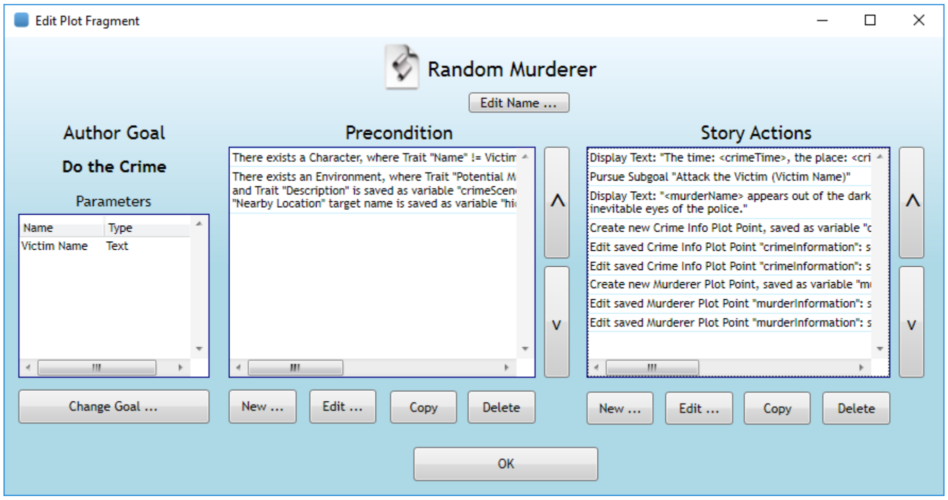
[3] J. Friedhoff. Untangling twine: A platform study. *Proceedings of DiGRA 2013: DeFragging Game Studies*, 2013.

[4] P. Gamallo. Dependency parsing with compression rules. *IWPT 2015*, page 107, 2015.

[5] A. K. Ingason, S. Helgadóttir, H. Loftsson, and E. Rögnvaldsson. A mixed method lemmatization algorithm using a hierarchy of linguistic identities (holi). In *Advances in Natural Language Processing*, pages 205–216. Springer, 2008.

[6] C. Klimas. Twine / An open-source tool for telling interactive, nonlinear stories. http://twinery.org/, 2009. [Online; accessed 22-November-2016].

[7] B. Li, S. Lee-Urban, G. Johnston, and M. Riedl. Story generation with crowdsourced plot graphs. In *AAAI*, 2013.



Author Goal: Do the Crime

Name	Type
Victim Name	Text

There exists a Character, where Trait "Name" != Victim

There exists an Environment, where Trait "Potential M
and Trait "Description" is saved as variable "crimeScen
"Nearby Location" target name is saved as variable "hi

Display Text: "The time: <crimeTime>, the place: <cri

Pursue Subgoal "Attack the Victim (Victim Name)"

Display Text: "<murderName> appears out of the dark
inevitable eyes of the police."

Create new Crime Info Plot Point, saved as variable "c

Edit saved Crime Info Plot Point "crimeInformation": s

Edit saved Crime Info Plot Point "crimeInformation": s

Create new Murderer Plot Point, saved as variable "m

Edit saved Murderer Plot Point "murderInformation": s

Edit saved Murderer Plot Point "murderInformation": s

Figure 9: Screenshot from the Wide Ruled 2 model

[8] Microsoft. .NET Framework. https://www.microsoft.com/net, 2002. [Online; accessed 13-December-2016].

[9] N. Montfort. Natural language generation and narrative variation in interactive fiction. In *Proceedings of the AAAI Workshop on Computational Aesthetics*, 2006.

[10] M. Owens and G. Allen. *SQLite*. Springer, 2010.

[11] J. Plisson, N. Lavrac, D. Mladenic, et al. A rule based approach to word lemmatization. In *Proceedings C of the 7th International Multi-Conference Information Society IS 2004*, volume 1, pages 83–86. Citeseer, 2004.

[12] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[13] D. V. Pynadath and M. P. Wellman. Generalized queries on probabilistic context-free grammars. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(1):65–77, 1998.

[14] J. O. Ryan, A. M. Fisher, T. Owen-Milner, M. Mateas, and N. Wardrip-Fruin. Toward natural language generation by humans. In *Proceedings of the INT*, 2015.

[15] J. Skorupski and M. Mateas. Interactive story generation for writers: Lessons learned from the wide ruled authoring tool. *Digital Arts and Culture 2009*, 2009.

[16] Unity. Unity 3D - Game engine. https://unity3d.com/, 2005. [Online; accessed 13-December-2016].

[17] A. Voutilainen. Part-of-speech tagging. *The Oxford handbook of computational linguistics*, pages 219–232, 2003.