ExpReal: a Writing Language and System for Authoring Texts in Interactive Narrative

Nicolas Szilas
TECFA-FPSE
Université de Genève
Genève, Switzerland
nicolas.szilas@unige.ch

Ruud de Jong
TECFA-FPSE
Université de Genève
Genève, Switzerland
ruud.dejong@unige.ch

Mariët Theune
Human Media Interaction
University of Twente
Enschede, The Netherlands
m.theune@utwente.nl

ABSTRACT

As interactive narratives, by definition, change according to the user's choices (dynamic story), so do the dialogue utterances by the characters. Writing all possible utterances manually faces scaling problems. This motivates the use of natural language generation techniques. We present ExpReal, a surface realizer and templating language that allows authors of interactive narratives to write flexible and enriched templates while maintaining control over their use. Templates are automatically selected based on author-specified conditions relating to the world state (e.g. characters' emotions) or the current task at hand. ExpReal has been developed to support at least three languages (English, French and Dutch).

CCS CONCEPTS

• **Computing methodologies** → **Natural language generation**; **Language resources**; *Discourse, dialogue and pragmatics.*

KEYWORDS

text generation, interactive narrative, templates, authoring

ACM Reference Format:
Nicolas Szilas, Ruud de Jong, and Mariët Theune. 2020. ExpReal: a Writing Language and System for Authoring Texts in Interactive Narrative. In *International Conference on the Foundations of Digital Games (FDG '20), September 15–18, 2020, Bugibba, Malta.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3402942.3402949

1 INTRODUCTION

Current adventure video games feature characters acting and talking with each other via vivid animations and well-crafted dialogues, but the user's influence on the story is limited [9, 14, 18]. Highly interactive narratives are computer-generated and react to the user, who plays the role of one of the main characters in the story. In the most interactive cases — dealt with by this paper —, both the global storyline and all the actions of the characters fully depend on the user's actions. Here, the generated events are calculated in real-time as a structure that would contain, for example, a type of action, a character doing the action, and objects involved in the action. These structures then need to be further processed to produce a text in natural language, serving either as textual description of the action or as characters' dialogue lines. This two-step generation process is common to very different approaches such as character-based planning [2], rule-based forward simulation of narrative acts [15], broad autonomous agents [6], and even machine learning based approaches [16]. Because the first step is the biggest challenge in the field of Interactive Narrative, the second step tends to be neglected. However, the quality of resulting text is essential in the final user experience. One cannot imagine a game such as the TellTales games without the colorful language used by the characters.

In the context of generated narratives, converting a structured description of an action into surface text requires a specific technological module called a 'surface realizer'. The quality of the final text depends on the technical characteristics of this module — the linguistic processing it can accomplish — but also on the human-crafted content that it takes as input: vocabulary, style, idiomatic phrases, etc. And finally, the quality of the texts depends on the ability of the linguistic module to let the authors express themselves with the tool. In this context, the surface realizer has to satisfy two constraints:

- Genericity: it must be able to cover with a limited number of abstract structures a large amount of specific realizations, in order to avoid writing a myriad of specific cases.
- Authorability: it must allow authors to express themselves by building, choosing and tuning well-designed sentences, with colored language, appropriate to the situation.

These two constraints are often contradictory: the second one is usually satisfied when an author works very closely with the concrete linguistic material, choosing the phrases and the words, while the first one asks for abstraction and generalization. Between the two extreme approaches that cover either one or the other constraint [12], that is classical Natural Language Generation techniques (based on linguistic processing) and canned text (whole sentence written for each distinctive case, with some empty slots to be filled when necessary) we propose a trade-off solution. ExpReal is a surface realizer and templating language that allows authors of interactive narratives to write flexible and enriched templates while maintaining control over their use.

2 RELATED WORK

SimpleNLG is a widely used surface realizer, originally for English [8], that has been adapted to multiple languages and offers users direct control over the realization process, including operations such as word ordering and inflection. It was designed as a simpler and more efficient alternative to other surface realizers, but its use

FDG '20, September 15–18, 2020, Bugibba, Malta

ACM ISBN 978-1-4503-8807-8/20/09…$15.00
https://doi.org/10.1145/3402942.3402949

still requires programming skills, which limits its accessibility to authors of interactive narratives. To make SimpleNLG more accessible, Caropreso et al. [1] created a graphical tool that allows authors to enter a sentence, indicate its variable parts, and select generation options such as pronoun gender and verb tense. SimpleNLG then generates a list of sentences resulting from all valid variable combinations. User tests showed that using the system was faster and more enjoyable than manual authoring of game texts.

Templates are commonly used for dynamic text production in games and interactive narratives. For example, character dialogue in Versu [7] is based on text templates marked up with their social and emotional effects. The interactive fiction (IF) system Curveship [11] allows authors to create templates with markers for the execution of various advanced linguistic functions. Unlike other, arguably more popular IF systems such as Inform 7 and TADS 3, Curveship uses a two-step generation process as described in Section 1. This allows for narrating the same events in different styles.

Another approach to 'authorable' text generation uses context-free grammars (CFGs): sets of rewrite rules that are recursively expanded. Expressionist [13] offers a graphical authoring interface for specifying CFGs with application-specific markers that can be used for generating in-game texts during game play. Improv [5] also uses grammars for in-game text generation, but Improv does not have a strict separation between simulation and text generation. Both Expressionist and Improv were inspired by Tracery [3], a popular tool for writing CFGs using a simple syntax.

ExpReal, presented in this paper, shares the aim of combining genericity and authorability with the systems presented above. ExpReal has most in common with the narrator component of Curveship [11] in its approach and functionality. Important differences are ExpReal's use of SimpleNLG in combination with templates, and its multilinguality: ExpReal currently works in three languages and can be extended to other languages with moderate effort.

3 REQUIREMENTS

The very first requirement of any surface realizer for interactive narrative is that it generates correct sentences. This typically corresponds to grammatical correctness, but can also be applied more loosely to colloquial language. In both cases, any breach in the language's norm would sound odd and must be avoided.

Although a surface realizer necessarily deals with some sort of processing (due to the genericity constraint), we introduce the key requirement that authors enter text (the authorability constraint). This text cannot be plain canned text, but should include additional information: annotations, tags, slots, formatting, etc. There is no first need for a custom authoring tool to enter this enriched text; authors can, for example, use a spreadsheet for that purpose.

A third requirement is variability. Contrary to adventure games, interactive narratives use the same action types repeatedly, either with the same variables or different variables. In both cases, if the system repeatedly generates the exact same expression, the user perceives it as mechanical and unnatural. For example, if a character invites another character to perform an action, she could initially say: "Marion, would you mind sitting down?" If this action is repeated, the user does not expect to read exactly the same sentence, but another formulation such as "Please, have a seat my dear."

When authoring text for interactive narrative, genericity and authorability often conflict: the author is not happy with a generic formulation and wants a specific more adequate text, which contradicts the genericity constraint. Solving this contradiction is the fourth requirement: the system should allow both generic and specific texts so the author can use a proper level of specificity.

Correctness, readability, variability and variable specificity constitute the four requirements for a surface realizer suited to interactive narrative. In the next sections, we will describe ExpReal, an **Exp**ressive **Real**izer designed to meet these requirements.

4 DESCRIPTION OF EXPREAL

4.1 Overview

The input for ExpReal consists of predicates, composed of a predicate symbol and a list of arguments: a general form that typically covers the actions and events that constitute a narrative. An example input is *AskHowToAchieve(Lili,Paul,BeAtOpera)*, corresponding to the action of Lili asking Paul how he plans to spend the night at the opera. Arguments are either generic or typed according to a list of predefined typical narrative-related elements: person, thing, place, task and goal. An argument can also contain a list of individual elements, coded as a string: "[e_1,e_2,...,e_n]". In addition to predicates, ExpReal also receives context information. The context contains the enunciation context, that is who is the speaker and who is the addressee, as well as application-specific information, such as the emotion of the speaking character.

As output, ExpReal generates a list of one or more texts (utterances). This is a list instead of just one string, because in some applications, the text corresponding to an action or event needs to be segmented in different parts. Our use cases involve embodied characters that may utter the first part of the text, then perform a specific gesture, and finally utter the second part of the text.

ExpReal is parameterized for each application with the authored data (the content) and the language to be used for generation. Three languages are currently covered: English, French and Dutch. The author specifies the content in a spreadsheet, with each row corresponding to a template set: an element to be transformed into text, as shown in Figure 1. All output languages are included in the same row in the spreadsheet, which simplifies translation and coherence checking. Templates can be written for elements of any size: sentences, parts of sentences, but also sequences of utterances by multiple characters. They can be nested, meaning that templates can be used to realize the variable parts inside other templates.

The next subsections explain how the authors would fill the spreadsheet, that is they detail the authoring language underlying ExpReal. We made an effort to make this language readable for non computer experts, and as close as possible to natural language.

	A	B	C	D	E
1	#Element	#Condition	#English	#French	#Dutch
2	AskToDo		can you $task?	pourrais-tu $task ?	kun je $task?

Figure 1: Authoring spreadsheet structure.

4.2 Template authoring

Templates are stored in a comma-separated values (CSV) file using semicolons (;) as separator, allowing for simple editing using Microsoft Excel, LibreOffice Calc or even simple text editors like Notepad. The input file requires five columns, see Figure 1. The first contains the names of the variables such as predicate types, character names and argument names. The second column is used for conditions (see Section 4.3). The last three columns contain the template in English, French and Dutch, respectively. These templates can include variables that are then replaced by the predicate's argument, person names or nested (sub)clauses.

A single template can contain multiple utterances from different speakers. We use em dash (—) or double en dashes (––) as markers to delineate the point where the speaker and listener swap roles.

4.3 Conditions

To fulfill the variable specificity requirement from Section 3, each template can be associated with a condition, entered in the condition column (see Figure 1). The condition specifies the cases in which the template can be used. Typically, one adds constraints on variable values. For example, the template `'|I could go to the circus with %Lili!|'` corresponds to the predicate *InformIntention(speaker, listener, task, goal)*, but only if the task is 'goToCircus'. This is ensured by the condition '`\$task=goToCircus`'.

The '=' and '!=' operators denote *equality* and *non-equality*. In case the argument is a list of elements, one can also test if the list contains an element or not by using the *containing* and *non-containing* operators '>' and '!>'. Several of these elementary conditions can be included in the same cell, separated by commas, to specify the conjunction (AND operator) of each elementary condition.

For a given predicate, there can be several template lines with various conditions, or without any conditions. Template selection is done as follows. First, all templates for other predicates are removed, as well as all templates with unmet conditions. Then, the template with the highest specificity (largest number of conditions) is chosen or one is randomly picked if multiple templates have equally high specificity. This randomness answers the requirement of variability, particularly when considering the case of multiple variables and embedded templates. Conditions can also be added for simple elements. For example, a character may be called Paul in general, but "dad" when the speaker is his son.

The specificity rule can be overridden using so-called 'overriding variables', prefixed with '@'. In one of our use cases (Section 5), an overriding variable is used to discriminate menu text from the actual utterance. We used the variable '@userChoice' and included the condition '@userChoice=true' for the menu texts, ensuring that these templates will always be selected for generating menu texts.

4.4 Linguistic Processing

The processing of the templates is performed in two steps: variable resolution and surface realization. In the first step variables are resolved, see Figure 2. This results in templates that consist of either the final text, or canned text enhanced with so-called 'grammatical blocks'. Grammatical blocks are slots within the template within { } braces. They are used to mark parts of the text that require further surface realization, such as inflection or pronominalization (replacing names or other referring expressions by pronouns such as "he" or "she"). Grammatical blocks contain a name and a value. The value is the variable or word requiring realization. The name is one of the grammatical roles according to the internal structure of the surface realizer, namely 'subject', 'verb', 'object', 'indirect object' or 'complement'. The value can range from a basic word (e.g. the verb "be", see Example 1) to a larger noun phrase (Example 2). In the latter example, the abstract formulation can result in various output sentences, depending on the context of enunciation, such as: "Olivia is with Julia tonight", "You are with Julia tonight", "She is with you tonight", "I am with her tonight", etc.

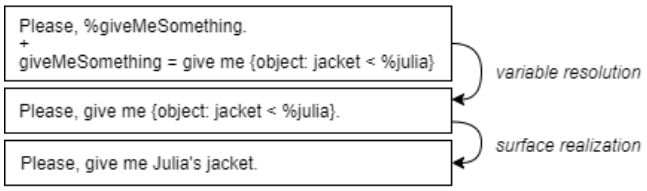

Figure 2: Steps of linguistic processing.

{verb: be} (1)

{subject:%Olivia} {verb:be} with {object: %Julia} tonight. (2)

Additional features of the grammatical blocks include ownership and adjectives. Ownership describes the factual ownership of the noun, e.g. "my jacket" vs. "Julia's jacket", see Example 3. Adjectives can be added in front of the noun or right after (especially in French, see Example 4). Grammatical blocks also implement a referring expression generator, which replaces the blocks with pronouns if doing so improves the natural flow of the dialog, according to the algorithm described by McCoy & Strube [10].

It's {object: big red jacket < %julia} (3)

C'est {object: la grande |veste| rouge < %julia} (4)

Surface realization, the second step in the linguistic processing resulting in the final output, is done with the help of SimpleNLG-NL [4], a trilingual extension of the bilingual SimpleNLG-EnFr [17], capable of handling English, French and Dutch. Its API is simple yet sufficiently powerful for our use cases. As ExpReal's templates are a mix of canned text and slots, SimpleNLG-NL is not used for determining word order, but only for inflection and other morphological transformations.

5 USE CASES

Currently, ExpReal is being used in two different applications, the Alzheimer Care Trainer and Feline. The following descriptions of these applications should provide an indication (though not a limit) of the types of practical use cases for the system.

The Alzheimer Care Trainer is a personalized 3D simulation of interactions with an Alzheimer's patient that can act as an online training tool for caregivers. The user (the caregiver) walks through scenarios they might encounter in real-life with the person they care for and can choose their own path through the interactive narrative. ExpReal is used to generate the text inside the characters' speech bubbles. This text depends on the personalization variables