Jl. of Interactive Learning Research (2012) **23**(3), 227-249

Template Authoring Environment for the Automatic Generation of Narrative Content

MARIA FERNANDA CAROPRESO, DIANA INKPEN,
FAZEL KESHTKAR, AND SHAHZAD KHAN
University of Ottawa, Canada
caropres@site.uottawa.ca
diana@site.uottawa.ca
akeshtka@site.uottawa.ca
shahzad@whyztech.com

Natural Language Generation (NLG) systems can make data accessible in an easily digestible textual form; but using such systems requires sophisticated linguistic and sometimes even programming knowledge. We have designed and implemented an environment for creating and modifying NLG templates that requires no programming knowledge, and can operate with a minimum of linguistic knowledge. It allows specifying templates with any number of variables and dependencies between them. It internally uses an existing sentence realization NLG tool in order to provide the linguistic background knowledge. We tested the performance and usability of our system in the context of interactive simulation games. We incrementally improved our system in order to obtain all the capabilities needed to reproduce all the sentences and templates manually created for already existing games. We trained the users and measured their satisfaction with the system by comparing the results of writing new games' narrative content manually vs. using our system. In general, the use of the system made the task faster, more enjoyable, and less prone to errors.

Introduction

Natural Language Generation (NLG) is the process of constructing outputs from non-linguistic inputs (Bateman, 2002) (Reiter and Dale, 2000). In other words, the role of NLG is to produce understandable text, from some nonlinguistic representation of information.

NLG is useful in systems in which verbal or textual interaction with the users is required, as for example Gaming, Robotics, and Automatic Help Desks. Using NLG systems instead of manually authored sentences would enable the software to adapt the expressed messages to the context of the conversation, and express past and future actions that may form this interaction.

However, the use of the available NLG systems is far from simple. The most complete systems often require extensive linguistic knowledge, as in the case of the KPML (KOMET-Penman MultiLingual) system (Bateman, 1997). A simpler system, SimpleNLG (Reiter, 2007), requires Java programming knowledge. This knowledge cannot be assumed for the content and subject matter experts who are members of the application development team. However, these individuals do need to interact with the NLG system in order to make use of the message generation capability to support their product development efforts. It is then necessary to provide them with an environment that will allow them to have access in a simpler way to the features they need of a specific NLG system.

We present an environment that provides simple access to the use of SimpleNLG in order to generate sentences with variable parts or templates. We developed this NLG Template Authoring Environment guided by the need of templates required for generating content for a digital-based interactive simulation game. The goal of this project was to provide the designers with an accessible tool they could use to create and manipulate the NLG templates, and thus generate sentences that would support the narrative progression of the game.

The NLG Template Authoring Environment asks for a model sentence and allows the user to mark the sections that are variable (i.e. dynamically generated), which would also serve to implicitly 'lock-down' the static elements of the generated sentences. Additionally, the content-author could then mark dependencies between variable elements. The System then displays a list of all the possible sentences that would be created from the given model with the specified variables and dependencies. After viewing this output, the user can refine the template model adjusting it to his/her needs.

The design and performance evaluation of the NLG Template Authoring Environment was guided by the requirements of a digital-based interac-

tive simulation game. A set of sentence templates covering different aspects were selected from the templates designed manually for that game. They were then recreated using our system, which has to be iteratively adapted until it was possible to implement all aspects of the templates. The usability of our system was then tested by comparing the performance achieved and the time required by the games' content writers to obtain the necessary sentences for two simple negotiation games, both manually and using the NLG Template Authoring Environment.

In the rest of this paper, we first introduce general concepts of NLG and some of the tools available. We then introduce Serious Games (or training games) and their need for NLG. With this we motivate the development of our NLG Template Authoring Environment and we describe its design and implementation. We evaluate its performance and expand the system capabilities to allow covering different aspects of the templates. We then evaluate the usability of our system by using it in the generation of textual content for two simple negotiation games. We finish the paper describing other systems similar to ours, and presenting our conclusions and future work.

NATURAL LANGUAGE GENERATION AND SIMPLENLG

As previously mentioned, the role of NLG is to produce understandable text from some nonlinguistic representation of information. The NLG process can be viewed as the inverse of Natural Language Understanding (NLU), as NLG maps from meaning to text, while NLU maps from text to meaning.

An NLG system will achieve its goal by performing different tasks such as selecting terminology and producing grammatically correct sentences. It will go through several stages in order to generate text which looks natural (similar to text that would be generated by a human being to express the given concepts).

According to (Dalianis, 1996), the stages of an NLG system are:

- content determination (choosing what concepts to express),
- lexicalization (choosing words to express the concepts),
- syntactic and morphological realization (producing the surface document or text by using syntactic and morphological rules),
- sentence aggregation (merging similar sentences into one sentence),
- referring expression generation (using pronouns to replace repeated noun phrases), and
- orthographic realization (resolving matters such as formats, casing, and punctuation).

There are two widely adopted approaches to NLG, the 'deep-linguistic' and the 'template-based' (van Deemter et al., 2005). The deep-linguistic approach attempts to build the sentences up from a logical representation. The template-based NLG systems provide scaffolding in the form of templates that contain a predefined structure and perhaps some of the final text. The 'deep-linguistic' approach to NLG is designed to be flexible and should be notionally able to express any sentence given a valid input logical form. Wide adoption of these systems has been constrained by the sophistication of the grammar system required, and the steep learning curve for the logical form. An example of this type of system is KPML.

In contrast to the flexibility of 'real' NLG systems, template based NLG systems are limited in the type of output they can generate as they are designed to operate with templates that must conform to a given structure. Due to the limited effort needed to create these systems, numerous examples exist, most of which are one-off developments. A commonly quoted example is that of the Forecast Generator (FOG) system designed to generate weather reports (Goldberg et al., 1994).

The 'deep-linguistic approach' is necessary in cases where there is no information available about the content or the form that the expressed text would take. This would be a requirement if NLG would be a component in a general purpose robot, as portrayed by Data in the popular Star Trek series. In these scenarios, there is likely to be very little in common between different instances of text generated by the system. In alternative scenarios, where the text generation will have a more homogeneous (and thus constrained) output, the simpler template based system is sufficient. Indeed, some have convincingly argued that both approaches can have similar levels of expressiveness if there is sufficient sophistication built into the template realization phase (van Deemter et al., 2005).

SimpleNLG (Reiter, 2007) is an sentence realization system that allows the user to specify a sentence by giving its content words and its grammatical roles (such as subject or verb). The specification can be presented at different levels of detail. For example "the black cat" could be specified as a noun phrase with no further details, or it could be specified as a noun phrase where "cat" is the head of the phrase, "black" is a modifier and "the" is the determiner.

SimpleNLG automates several tasks, such as orthography, morphology, and grammatical realization. For the latter, it uses grammar rules to convert abstract representations of sentences into actual text. SimpleNLG also permits the user to specify several features for the main verb, such as: tense (present, past or future); whether or not it is subjective, progressive, passive

or perfect; whether or not it is in interrogative form; whether or not it is negated; and which, if any, modal to use (i.e. could, must).

SimpleNLG is implemented as a Java library and it requires java programming knowledge to be used. It allows the user to define flexible templates by using programming variables in the sentence specification. The variable parts of the templates could be filled with different values. When templates are used without an NLG system, they are called canned-text, and they have the disadvantage of not being very flexible, as only the predefined variables can change. When templates are defined using SimpleNLG, they keep all the functionality of the NLG system (for example, being able to modify the verb features or the output format, and making use of the grammatical knowledge), while also allowing for the variable values to change.

SERIOUS GAMES AND THE NEED FOR NLG

The term 'serious games' refers to a sub-category of interactive simulation games in which the main objective is to train the player in a particular subject matter. The player is typically presented with challenging situations and is encouraged to practice different strategies at dealing with them, in a safe, virtual environment. Through tips and feedback provided during and at the end of the game, the player develops an understanding of the problem and what are the successful ways of confronting it (French et al., 1999).

As an example of a serious game, we briefly describe DISTIL's game ISO 14K. The objective of this game is to train the player in the process of implementing an environmental management system (EMS). The player controls the main character of the game, who manages the implementation of a standards-based process in a simulated fictional organization. S/he is responsible for hiring more employees, as needed, and assigning each of them different tasks to perform. All the other characters of the game are controlled by the computer. The player will constantly make decisions that will result in a successful or unsuccessful implementation of the process. In either case, the objective of the game would be reached, as the player would have acquired new knowledge that would hopefully be useful when dealing with a real situation.

Serious games are generally content oriented and a significant amount of information is provided to the player through images, sounds and narrative. In many cases, the narrative is incorporated in the game through dialogues or other forms of interaction between game characters. In the game that we used, for example, the narrative is provided as e-mail messages from other characters to the main character.

The considerable amount of textual information required in serious games can be a burden on the game designers. When the information is incorporated interactively, it is desirable that it simulates an exchange realized by humans (Lin & Kraus, 2010). Given the many possible game scenarios and situations arising from the player decisions, manually writing the information exchanges can account for many months of work, as there are changes required during each iteration of the game in order to keep the feedback consistent with the updated narrative. It is then necessary to include templates that provide the basic information, combined with variable parts that adapt the narrative to the circumstances. Other automated methods also attempted to generate such adaptive content (Rowe et al, 2008; Kenny et al, 2007; Méndez & Nakayama, 2008).

The template approach to textual information generation was first tried by DISTIL in the game ISO 14K, while the manual approach was used in a previous version of a similar game, ISO 9K. By employing templates, the narrative was more efficiently produced and more sophisticated. While in the previous version of the game the feedbacks were directly tied to the failure or success of the actions, in the current version of the game, the feedbacks were systematically generated for each character and task combination available to the player. With the use of templates, the feedback available was also made scenario specific in the current version of the game, and thus more useful information was available to the player. The development time was reduced significantly, mainly when the same templates created for the ISO 14K game were re-used for a new version of a similar game, ISO 18K. The development time and the number of textual feedbacks generated for each game are shown in Table 1.

Table 1
Development Time

Game	Feedbacks	Time
ISO 9K (Manual)	200	6 months
ISO 14K (Templates)	8,142	4 months
ISO 18K (Reuse)	8,542	one week

The following example shows a template used in the game ISO 14K. In it, PRONOUN_SUBJECTIVE would return either *I* or *we* depending on the ACTOR, PRONOUN_POSSESSIVE would return either *my* or *our* depend-

ing on the ACTOR, DEPARTMENT would take a value from a list of the company departments and representatives depending on the ACTION.

PRONOUN_SUBJECTIVE(ACTOR)
felt competent to do this job because of
PRONOUN_POSSESSIVE(ACTOR)
knowledge of DEPARTMENT(ACTION).

Using the template in the previous example, sentences like the ones given below could be generated for actors that represent individuals (the first sentence) or for actors that represent groups (the second sentence):

- I felt competent to do this job because of my knowledge of the HR/ Training Department.
- We felt competent to do this job because of our knowledge of the Operations Department.

Because both PRONOUN_SUBJECTIVE and PRONOUN_POSSESSIVE depend in this case on the same ACTOR, there is inter-dependence in the values they can take. This type of dependency is the one we refer to in the following sections.

The above templates were hard-coded in the game ISO 14K. In our current work, we propose the use of a more flexible way of generating templates for the dialog of the games. We present our System, NLG Template Authoring Environment, which takes advantage of the grammatical knowledge of SimpleNLG in a simpler way. It does not require the user to have either advance linguistic or programming knowledge. We used the templates in the game ISO 14K as a guideline of the minimum capabilities that our system should provide.

NLG TEMPLATE AUTHORING ENVIRONMENT

With the objective of permitting the game designers to study the sentence templates they would propose for the games, we have come up with the idea of providing a Natural Language Generation Template Authoring Environment. In the context of creating sentence templates for games design, this system bridges the gap between the game designers' content knowledge and the knowledge required for the use of NLG systems.

This environment allows the user to give an example sentence, to define what parts would be variable and what would be the possible values, and to specify dependencies between variables. It then shows the user all the pos-

sible sentences that could be generated from the given template by calculating all the possible combinations of variable values that respect the specified dependencies. The user can then refine the template by changing either the given example or the specified variables and dependencies, in order to adjust the generated sentences to the needs of the game.

DESIGN

A graphical design for the NLG Template Authoring Environment is shown in Figure 1. This also shows a simple example of a sentence with three variables and a dependency specified.

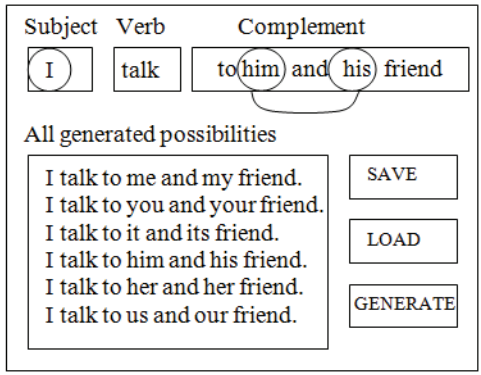

Figure 1. NLG Template Authoring Environment

As shown in Figure 1, the system allows the user to input an example sentence with an identified main verb, a subject, and a complement (see the text in the respective boxes). In addition, information for the verb (i.e., tense, form, modals) could be specified. By default the present tense, non-progressive form, active voice will be used. The user has the choice of either changing these options or adding new options.

The system allows the user to identify variables in the subject and the complement of the sentence (see the ovals around some of the words entered in the text boxes). For each of the specified variables, the user has to indicate its type (i.e., personal pronoun, possessive pronoun, Employee_ type) and which values of that type are allowed (i.e., all personal pronouns, or only "she" and "he"). The user can also indicate dependencies between variables (see the arc linking the variables containing "him" and "his" in the example).

All the information provided to create a template (the example sentence, its variables and dependencies) can be saved and recovered later on through the provided utilities SAVE and LOAD. Through the use of the GENERATE utility, new sentences that follow the template indicated in the example sentence are generated and displayed back to the user. The displayed sentences are the result of combining the values of the variables and the verb options (when more than one was specified) in all possible ways while respecting the dependencies between variables.

IMPLEMENTATION

The NLG Template Authoring Environment has been implemented in Java. The SimpleNLG library was used to automatically generate correct sentences and provide the user with the possibility of exploring different attributes to the verb.

The variables are represented by objects which store all the necessary information, such as: variable type, default value, current value, gender and number of the current value, and other information that is used when generating all possible combinations. The variable type refers to a text file containing all the possible values with their respective syntactic information (person, number and gender) which will be used for agreement with the verb and for dependency between variables.

Once a combination of values for all the variables is generated and considered as a valid choice (after filtering according to the dependencies), the static and variable parts of the sentence are reunited and provided to methods that use the SimpleNLG package in order to realize the sentences. At this stage, all required modifications to the verb are performed, and several possibilities could be displayed according to the user choice for the verb options. For example, if the user has indicated present, past and future as the verb tense options, three sentences (one realizing each tense) will be displayed for the current combination of variable values.

INTERFACE

A user-friendly intuitive graphical interface has also been implemented in Java using the Swing library. A partial screenshot of this interface is shown in Figure 2.

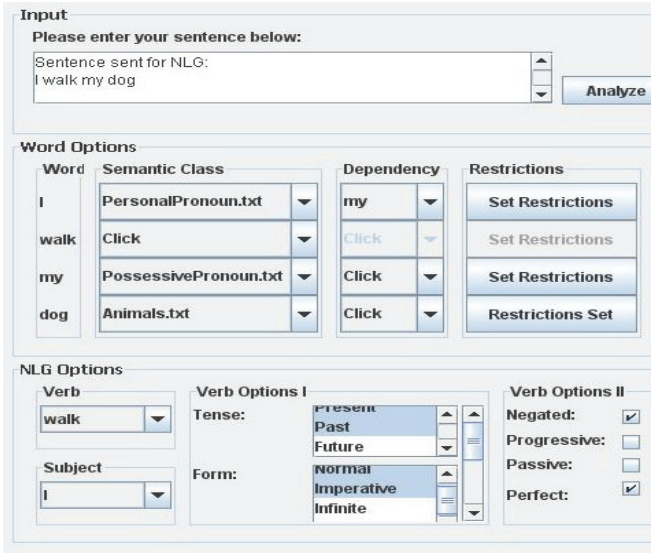

Figure 2. Graphical Interface

When using this interface, the user first enters an example sentence and clicks on Analyze. Next the user indicates that a section is variable by giving a type or semantic class to the word in that section. As previously mentioned, the values of a semantic class are stored in a text file, which allows the user to create new semantic classes as needed. Restrictions to the values that a variable can take are also indicated through the graphical interface. Dependencies can be indicated only between already declared variables. The main verb and all its options are indicated in the section at the bottom of the graphical interface.

In the partial screenshot shown in Figure 2, the example sentence is "I walk my dog", "I" is a variable of type Personal Pronoun, "walk" is the main verb, "my" is a variable of type Possessive Pronoun, "dog" is a variable of type Animals and there is a dependency between "I" and "my" (which will allow to make their values agree in person, number and gender when generating all possible combinations).

In Figure 2, the user has selected the values "present and past" for the verb tense and "normal" and "imperative" for the verb form. Therefore, four sentences will be generated for each combination of the variables' values (one sentence for each combination of the tense and form selections). These sentences will have the verb negated and will use the perfect tense (as indicated by the verb options in the last column).

TESTING THE SYSTEM'S CAPABILITIES

In order to verify the correct functioning of the NLG Template Authoring Environment, we selected a set of sentence templates from the game ISO 14K. The templates were selected manually, keeping in mind the need to cover different aspects, as for example the number and type of the variables and dependencies. The testing of these examples covers for many more templates of the same type. The five selected sentence templates that form our testing set are displayed in Table 2.

Table 2
Testing examples

Ref. number	Template
1	The ACTORS (ME/US) could help DEPARTMENTS.
2	The ACTORS IS/ARE now available to help.
3	I/WE struggled because of MY/OUR lack of knowledge.
4	I/WE AM/ARE pleased to report that I/WE completed the task TASKS.
5	I/WE WAS/WERE not the greatest choice for keeping things moving along quickly.

In these template examples, we show in capitals the variable parts of the templates. ACTORS, DEPARTMENTS and TASKS refer to one of several possible nouns previously defined for each of the classes with those names. The terms in capitals separated by a "/" already display all the accepted values for that variable (for example I/WE represent a variable of type personal pronoun which could take only the selected values "I" or "we" and the rest are filtered out).

The goal of the tests that we performed was to verify that the system provides the minimum capabilities expected. Therefore, when problems were discovered, we improved the system in order to successfully produce the selected templates. Details of these technical issues and modifications can be found in our previous publications.

TESTING THE SYSTEM'S USABILITY

In order to test the Template Generation Authoring System's usability, we have trained three users. We gave them an introduction to templates and the system's general goal. We explained the meaning of semantic classes, variables, and dependencies. We described the different generation options that could be passed to SimpleNLG and what changes would they produce on the resulting sentences. We finally showed them the interface and created some example templates together. This whole training took around an hour, after which they were able to successfully create and iteratively refine their own templates. After allowing them to experiment with the system for a couple of days, they were asked to complete the evaluation questionnaire found in Appendix C. According to their answers, they found the system easy to use and they only needed a day to familiarize themselves with the interface options and to feel comfortable using it.

For the purpose of further testing the usability of our System, we have designed two simple negotiation games. The first game consists of negotiating the sale of a house from either the buyer or the seller perspective (scenario one and two, respectively). The second game consists of negotiating the salary and benefits of a job offer, from either the applicant or the employer perspective (again, scenario one and two, respectively).

NEGOTIATION GAMES

The objective of a negotiation game is to train the player in being able to recognize and react to offers being made. The exchange of information between the parties involved in the negotiation is limited to a certain number of turns. The issues and values to be considered vary depending on the specific game and the selected perspective (the chosen scenario).

The goal of both parties engaged in the negotiation is to obtain the best possible advantage from their respective perspective (i.e., in negotiating the sale of a house, the seller would like to sell it at the highest price, while the buyer would like to buy it at the lowest price.)

As an example, we now consider the game of negotiating the sale of a house and the scenario in which the player is the buyer and the computer system is the seller. The information to be exchanged by the parties could include: the price, the closing date, whether appliances would be included or not, and whether certain conditions (such as a mortgage approval) should be satisfied first.

In this game, for this scenario, the player (buyer) can make offers to purchase the house by choosing different values to the previously mentioned information. The computer system (seller) will reply to an offer by either accepting it or rejecting it. In the latter case, it will provide feedback on the reasons for the rejection. The player will then re-adjust the offer until it is accepted, or the maximum number of exchanges is reached.

The decision made by the computer system on whether to accept or reject an offer is based on a simple threshold function. For the different possible values of the information included in the offer, the computer system attributes a score that represents its contribution to a successful sell. For example: the minimum sell price will have a neutral contribution to the final sale decision and therefore have a score of 0; a price higher than the minimum will contribute positively to the sale and therefore have a positive score; and a price lower than the minimum will contribute negatively to the sale and therefore have a negative score. When an offer is made the computer system will add the contribution scores of all the information included in the offer and compare the result with a pre-established threshold. The offer will be accepted if the result is higher than the threshold. In order to make the game interesting, the threshold value and the information scores are unknown to the player.

When considering the same game on scenario two (the player is the seller and the computer system is the buyer), the computer system will be making the offers and the player will be accepting or rejecting them and providing the respective explanations.

NEGOTIATION GAMES' CONTENT GENERATION EVALUATION

We evaluated the usability of the system by creating the textual content of the negotiation games described in the previous sub-section. We were interested in comparing the performance when writing the phrases manually and when using our Templates Generation Authoring System. For this reason, the content generating task was done manually for the first scenario, and using the system for the second scenario, for both games.

The content requirement for the games was explained to the content writers and they were asked to produce all the text needed. They were provided with examples of the possible phrases and the format in which they were required. The description of the games and the tasks provided to them are shown in Appendices A and B.

The content writers were assisted by the system's trainer during the generation of the sentences for the first game. They used the system unas-

sisted for the second game. The time invested and the errors made by the users when generating content for the negotiation games are shown in Table 3.

Table 3
Errors Made and Time Invested (minutes) when Generating Content for the Negotiation Games

	User 1		User 2		User 3	
Game 1 Manually	1 spelling error (repeated 4 times)	40	1 spelling error (repeated 40 times)	40	4 spelling errors (repeated 99 times) 2 missing sentences 4 repeated sentences	32
Game 1 System	1 un-IDed verb 1 missed filter	21	4 missed filters	31	1 spelling error 4 missed filters	29
Game 2 Manually	no errors found	25	1 spelling error (repeated 40 times)	17	3 spelling errors (repeated 55 times) 2 extra sentences	35
Game 2 System	1 spelling error	20	1 template missing	12	1 spelling error 1 missing template 2 extra templates 2 wrong semantic classes	15

In general, the writers took less time to create the templates using the system (that will generate all the necessary sentences, from the created templates) than it took to create all the sentences manually. As these three writers wrote the manual sentences first, it could be the case that the speed up resulted from thinking through and becoming familiar with the sentences generated for the game. However this hypothesis was discarded when a fourth content writer was asked to perform the tasks for the same two games but creating first the templates and then the manual sentences. Using the system first still resulted in shorter times. These results are shown in table 4.

Table 4
Errors Made and Time Invested (minutes) when Generating Content for the Negotiation Games (user 4 created the sentences using the system first)

	User 4	
Game 1 Manually	1 spelling error (repeated 4 times) 3 missing verbs	29
Game 1 System	no errors found	22
Game 2 Manually	1 spelling error (repeated 25 times) 1 wrong sentence (repeated 11 times)	46
Game 2 System	extra information in 6 templates	30

For the four content writers, some errors were introduced by the use of the system, the most common being forgetting to define a filter. This error will result in the generation of extra sentences that were not required and that will never be used by the game. It will not affect the quality or the availability of the required sentences.

Spelling errors were less common when using the interface. In addition, spelling errors that appear in the manually generated sentences repeat themselves many times given the tendency of the content writers to copy-and-paste text. Correcting the manually-generated spelling mistakes will imply re-writing the word each time the error was repeated. Whereas correcting the spelling mistakes in the templates will require only one manual intervention and the automatic re-generation of all the sentences. This is also true for any change required in the sentences due to the iterative refinement of the narrative.

The content writers answered an evaluation questionnaire after generating the text for the negotiation games. The questionnaire is shown in Appendix D. According to their answers, using the system to create the templates was “more enjoyable” and “less of a headache” than manually writing all the sentences. This was even more evident during the evaluation with the second game. While manually-generating the sentences for the second game, the writers re-used the sentences from the first game, many changes were involved and the process was still time consuming. When using the system, the writers re-used the semantic classes from the first game, adapting them as needed. The new templates were rapidly created given the previous similar experience.

COMPARISON WITH OTHER SYSTEMS

In this section we present other systems that, given some visual and philosophical similarities in the provision of a point-and-click interface for novice users, might seem closely related to ours. The difference between these systems and ours are explained.

WYSIWYM SYMBOLIC AUTHORING SYSTEMS

WYSIWYM (What You See Is What You Meant) is a natural language based technique used to create and update objects in knowledge bases (Power and Scott, 1998). It has been used in Symbolic Authoring Systems that allow the user to create symbolic representations from which documents in different languages can be generated.

Symbolic Authoring systems implemented using the WYSIWYM technique provide the user with an interface that describe in natural language the content of a knowledge base (i.e., which data objects are contained in it and what is their current completeness status). They also allow the user to add new data objects or edit already present ones in order to complete general sentences in the displayed text. Each time the knowledge base is updated through this process a new text that reflects its current state is generated and displayed. The final product of the process will be the desired document generated from the resulting knowledge base. By using the WYSIWYM technique, the Symbolic Authoring systems are accessible to users who are not experts in knowledge representation or computational linguistics.

It must be noted that the general sentences (or templates) that are completed through the use of the interface have to be embedded in the system, and therefore a new system has to be generated for each application.

The interface of these systems and the fact that options are selected from pop up menus to complete sentences according to different object types can make them look similar to our system. However the goal and final product of our system are very different. The goal of our system is for the user to design templates from scratch, which makes them domain independent. By looking at all the possible sentences that could be generated from a given template, the user can refine the template in order to obtain all and only the required sentences for a specific need. The final sentences produced by the template (generated through this process) are made available to the user. These sentences are then incorporated in the narrative of the many different events that follow possible scenarios and actions taken in a digital game.

NATURAL LANGUAGE MENUS (NLMENUS)

NLMenus (Tennant et al., 1983) is a concept used in some systems that allow the user to access resources by asking for information using natural language, such as consulting an airport database. The use of NLMenus allows users to be 'guided' to valid queries without extensive training.

Instead of letting the user express any possible query that the system might not understand, with NLMenus the user is restricted to ask only questions that follows the systems internal grammar. This is realized through a system-initiated process that facilitates the creation of NL queries. In each step, options to complete the query are presented to the user in popup menus. According to the choices made, the query is extended and new options are made available if appropriate. Internally, all possible parse trees are typically generated and kept track of during the query generation process.

The grammars used by these NLMenus are restricted to the system's topic and desired queries, and only those queries that the system will be able to reply to are allowed to be generated. Even the vocabulary used is restricted by the grammar.

In contrast, our system uses the general English grammar from SimpleNLG. The sentences created using our system are therefore much less restricted by the grammar, and the vocabulary itself has no limitations. Even the values of the semantic classes used for the template variables are specified by the user.

CONCLUSIONS AND FUTURE WORK

We have identified the need for an NLG Template Authoring Environment that allows game content designers without linguistic and programming background to experiment with and finally design language templates.

We have designed a system that allows the user to specify an example sentence together with variables, its dependencies, and verb options that complete the template. This system shows the user all the possible sentences that could be generated with the specified template. It can be used to refine the template until it satisfies the user's needs.

We have implemented a system that makes use of the SimpleNLG java library which provides us with correct sentences and the possibility of including many verb variations, such as tense, form and modals.

We have evaluated the capabilities of our NLG Template Authoring Environment in a set of sentence templates from a digital-based interactive simulation game that covered different characteristics.

We have also provided the system with a user-friendly intuitive graphical interface that allows the user to iteratively make changes to the sentence, variables and dependencies definitions, and to set and modify the verb options. In the future we will extend this interface in order to allow the user to create new semantic classes without having to manually edit the text files. We plan to also offer the user a syntactic analysis of the example sentence and suggestions for the type of variables to be used.

The convenience of using this interface was evaluated in the context of the development of two negotiation games. The games' content writers were trained in the use of the system, were introduced to the negotiation games concepts, and were provided with specifications for the sentences that needed to be generated. They wrote the sentences for different scenarios of the game either manually or using the system. The time spent in the tasks and the quality of the obtained outputs were compared. In general, the use of the system made the task faster, more enjoyable and less prone to errors.

Current research has attempted to employ the ability to generate interaction content to support the illusion of emotions (Strong et al., 2007; Stearn, 2002; Alexandrova et al., 2010). We are studying the possibility of using our system in the generation of feedbacks that account for the characters' personality and mood. With this in mind, we are using machine learning in order to generate lists of expressions that denote formal/informal and friendly/unfriendly discourse. The words in these lists could be used as variables of the respective semantic class to change the tone of the generated feedback. A new type of agreement will have to be implemented for these classes.

References

Alexandrova, I.V., Volkova, E.P., Kloos, U., Bülthoff, H.H. & Mohler, B.J. (2010) Virtual Storyteller in Immersive Virtual Environments Using Fairy Tales Annotated for Emotion States. Proceedings of the Joint Virtual Reality Conference of EuroVR.

Bateman, J.A. (1997). Enabling technology for multilingual natural language generation: the KPML development environment. Journal of Natural Language Engineering, 3(1):15-55.

Bateman, J.A. (2002). Natural Language Generation: an introduction and open-ended review of the state of the art.

Dalianis, H. (1996). Concise Natural Language Generation from Formal Specifications, Ph.D. Thesis, Department of Computer and Systems Sciences, Royal Institute of Technology, Stockholm University. Report Series No. 96-008, ISSN 1101-8526, SRN SU-KTH/DSV/R 96/8 SE.

van Deemter, K., Krahmer, E. & Theune, M. (2005). Real versus Template-Based Natural Language Generation: A False Opposition? In Computational Linguistics, 31(1): 15-24.

French, D., Hale, C., Johnson, C. & Farr, G. (1999). Internet Based Learning: An introduction and framework for higher education and business. London, UK: Kogan Page.

Goldberg, E., Driedger, N. & Kittredge, R. I. (1994). Using Natural Language Processing to Produce Weather Forecasts. IEEE Expert: Intelligent Systems and Their Applications. 9(2): 45-53.

Kenny, P., Hartholt, A., Gratch, J., Swartout, W., Traum, D., Marsella, S. & Piepol, D. (2007) Building Interactive Virtual Humans for Training Environments. Proceedings of the Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)

Lin, R. & Kraus, S. (2010) Can automated agents proficiently negotiate with humans? Communication of the ACM. 53 (1): 78-88.

Méndez, D.J.R., Nakayama, K. (2008) Adaptive Self-feeding Natural Language Generator Engine. Proceedings of Intelligent Virtual Agents, 533-534.

Power, R. & Scott, D. (1998). WYSIWYM: Knowledge Editing with Natural Language Feedback. Proceedings of the 13th Biennial European Conference on Artificial Intelligence (ECAI-98).

Reiter, E. & Dale, R. (2000). Building Natural Language Generation Systems (Studies in Natural Language Processing), Cambridge University Press.

Reiter, E. 2007. SimpleNlg package: http://www.csd.abdn.ac.uk/ereiter/simplnlg

Rowe, J.P., Ha, E.Y. & Lester, J.C. (2008) Archetype-Driven Character Dialogue Generation for Interactive Narrative. In Intelligent Virtual Agents, Springer.

Stearn, A. (2002) Creating Emotional Relationships with Virtual Characters. In Emotions in humans and artifacts, Robert Trappl, Paolo Petta, Sabine Payr (eds.). MIT Press.

Strong, C., Mishra, K., Mehta, M., Jones, A. & Ram, A. (2007) Emotionally Driven Natural Language Generation for Personality Rich Characters in Interactive Games Proceedings of the Third Conference on Artificial Intelligence for Interactive Digital Entertainment (AIIDE-07), Stanford, CA.

Tennant, H. R., Ross, K. M., Saenz, R. M., Thompson, C. W. & Miller, J. R. (1983). Menu-Based Natural Language Understanding. Proceedings of the 21st annual meeting on Association for Computational Linguistics.

APPENDIX A

House Purchase Negotiation Game Content Generation

Background:
A negotiation game is being created to train the player in being able to recognize and react to offers being made. It is designed to focus on the issues, and purchase offers only.

House Purchase Offers
- Price [260000, 270000, 280000, 290000, 300000]
- Closing Date [2 weeks, 4 weeks, 6 weeks, 8 weeks]
- Appliances [included, not_included]

Rejections Reasons:
- Price: Too_low, no_problem
- Closing: Too_early, no_problem, too_late
- Appliances: included, no_problem

Scenario 1: (create manual output)
Purchase of a house [Player is the buyer; Computer AI is the seller]
You have offered to purchase the house for 260000, with appliances included, with a proposed closing date of two weeks from now.
Congratulations, you are the new owner. The closing is in four weeks, the appliances are not included, and it costs you 300000.
Your offer was rejected. The seller said that the price was too_low and the appliances were included.

Scenario 2: (create templates using tool)
Sale of a house [Player is the seller; Computer AI is the buyer]
An offer was made to buy the house for 260000, with appliances included, with a proposed closing date of two weeks from now.
Congratulations, you have sold your house. The closing is in four weeks, the appliances are not included, and they paid you 300000.
You have rejected the offer because the price was too_low and the appliances were included.

Example of generated Game XML:
<Sent price: 260000, closing: 2 weeks, appliances: included>
You have offered to purchase the house for 260000, with appliances included, with a proposed closing date of two weeks from now. </Sent>

<Sent price: 260000, closing: 2 weeks, appliances: not included>
You have offered to purchase the house for 260000, with appliances not included, with a proposed closing date of two weeks from now. </Sent>
. . .
<Sent price: 300000, closing: four weeks, appliances: not included>
Congratulations, you are the new owner. The closing is in four weeks, the appliances are not included, and it costs you 300000. </Sent>
. . .
<Sent price: too low, closing: no problem, appliances: included>
Your offer was rejected. The seller said that the price was too_low and the appliances included. </Sent>

APPENDIX B

Job Offer Negotiation Game Content Generation

Note: Outputs and Templates from scenario 1 and 2 may be reused as this also involves similar game actions available to players
Job Offers:
- Salary [60K, 70K, 80K, 90K, 100K]
- Start Date [2 weeks, 4 weeks, 6 weeks, 8 weeks]
- RRSP Co-payment [included, not_included]
Rejection Reasons:
- Salary: Too_high, no_problem
- Start Date: Too_early, no_problem, too_late
- RRSP Co-payment: included, no_problem

Scenario 1: (create manual output).
Negotiate salary and benefits for new job [Player is the job seeker; Computer AI is the employer]
You have offered to join the firm for 100K, with RRSP Co-payment included, with a proposed start date of two_weeks from now.
Congratulations, you obtained the job. The start date is in four_weeks, RRSP Co-payment is not included, and you have a salary of 60K.
Your offer to join the firm was rejected. The employer said that the salary was too_high and that the RRSP Co-payment was included.

Scenario 2: (create templates using tool)
Negotiate salary and benefits for new job [Player is the employer;

Computer AI is the job seeker]
You offered the candidate to join the firm for 100K, with RRSP Co-payments included, and a proposed starting date of two weeks from now.
Congratulations, the candidate accepted the job. The starting date is in four weeks, RRSP Co-payment is not included, and the salary is 100K.
You rejected the candidate's offer to join the firm because the salary was too_high and the RRSP Co-payment was included.

APPENDIX C

Templates Generation System Evaluation Survey

(answered by content writers after training).
After receiving training, is the interface easy to use? 1 (very complicated) - 5 (very easy)
How long did it take you to learn to use the interface? 1 hour - 1 day - 1 week - 1 month
How long did it take you to feel comfortable using the interface? 1 hour - 1 day - 1 week - 1 month
How clearly are options presented in the interface?
1 (not clear) - 5 (very clear)
Indicate the type of sentences you have created:

- short sentences / long sentences
- one subject / several subjects
- one verb / several verbs
- one variable / several variables
- with many/few dependencies / without dependencies
- facts / negations / questions
- progressive / passive / perfect
- form: normal / imperative / infinitive
- verb agreement: default / plural / variable

With respect to the generated sentences:
are them always correct? In which types (from above) they were not correct? Please give examples of errors in generation.
Is there any other type of sentences you would like to create?
How useful do you consider the system?
1 (not really useful) - 5 (very useful)
Please explain why you do/don't consider the system useful.
What would make the System more useful?

APPENDIX D

Templates Generation System Evaluation Survey

(answered by content writers after using it for generating the content for scenario 2 of both negotiation games.)
Advantages and disadvantages of writing the sentences manually.
Advantages and disadvantages of using the System.
Advantages and disadvantages between:
a) creating all the different rejection templates
b) having all the combinations already in a semantic class and using it for generating only one rejection template.

Acknowledgment

We thank Carrie Lavis for her insights on the negotiation games; Waqqas Abdul-Basit for programming the visual interface; and the system's users for helping us on testing the system.