Automatic Generation of Narrative Content for Digital Games

Maria Fernanda CAROPRESO[1], Diana INKPEN[1], Shahzad KHAN[2] and Fazel KESHTKAR[1]
[1] School of Information Technology and Engineering, University of Ottawa,
Ottawa, ON, K1N6N5, Canada
emails: {caropres, diana, akeshtka}@site.uottawa.ca
[2]DISTIL Interactive, Ottawa, ON, K2H 8R6, Canada
email: srkhan@gmail.com

Abstract:

Interactive simulation games used for training usually require a large amount of coherent narrative content. An effective and efficient solution to the narrative content creation problem is to use Natural Language Generation (NLG) systems. The use of NLG systems, however, requires sophisticated linguistic and sometimes programming knowledge. For this reason, NLG systems are typically not accessible to the game designers who write narrative content. We have designed and implemented a visual 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 automatically generates all the sentences that follow the created template. It uses SimpleNLG to provide the linguistic background knowledge. We tested the performance of our system in the context of an interactive simulation game.

1. 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 systems are 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.

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 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 development team. However, these individuals could benefit from the message generation capability of NLG systems to support their product development efforts.

We have developed a system 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 game content 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. This is a critical requirement of games, as there are many potential paths through the games and therefore the possible state-space is usually very large.

The NLG Template Authoring Environment enables content writers to provide an example sentence, and leverage NLG to expand it to a large set of content. It asks the user to mark the sections of the example sentence that are variable (i.e. dynamically generated)[1]. Additionally, the content-author could then mark dependencies between variable elements. The system output template rules[2], and 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 her needs.

The rapid adoption of information retrieval tools has been supported by the availability of an iterative process of identifying the information need of the user via the progressive refinement of supplied keywords, in response to the display of search results. Our hope is that a similar process would hasten the adoption of NLG tools by the wider information processing community as well. We are developing a similar process that can support the NLG system users to iteratively improve the templates that they create. The key value provided by this process is that the template is created in a natural manner, without 'leap-of-faith' symbolic expressions. The symbolic expressions are created behind the scenes based on the user choices. Thus, non-experts can iteratively improve their template in a 'try-and-see' iterative fashion without needing to build a complete theoretical representation of the template before

[1] This would also serve to implicitly 'lock-down' the static elements of the generated sentences.
[2] These are not shown to the user, but are available in an XML format for examination and (if necessary) editing by experts.

seeing any results. Although experts may still prefer to approach the NLG template generation task using more powerful symbolic representation, this system is very valuable for the software developers, novice creative writers and industrial engineers who would not be sufficiently motivated to develop NLG template writing skills in order to employ automatic generation as a part of their system. Thus, this NLG Template Authoring Environment has been designed to be a 'disruptive innovation' that enables individuals who would not normally be able to use this technology, due to the specialized skills required to do so (Christensen and Raynor, 2003).

The design and performance evaluation of the NLG Template Authoring Environment was guided by the requirements of a digital-based interactive 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 all aspects of the templates were possible to implement.

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 developing of our NLG Template Authoring Environment and we describe its design and implementation. We then evaluate its performance and expand the system capabilities to allow covering different aspects of the templates. We finish the paper presenting our conclusions and what is pending as future work.

2. 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).

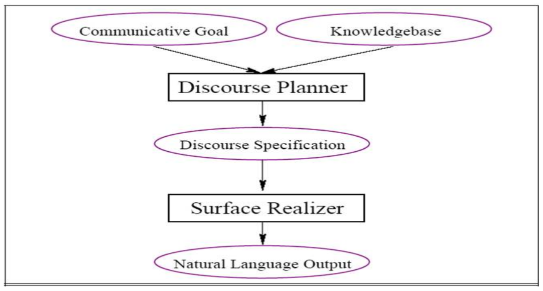


Figure 1. Architecture of NLG-systems

Dalianis also introduced the NLG Architecture System shown in Figure 1, where:

- the Discourse Planner chooses the appropriate content to express the communicative goal and gathers information from a knowledgebase to transform the content into text; and
- the Surface Realizer generates sentences for the discourse specification based on its lexical and grammatical resources.

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 wholly 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 system, 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 system, 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' system is necessary in cases where there is no information available a priori about the content or the form that the expressed text would take. This would be a requirement if the NLG system 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).

2.1 SimpleNLG

SimpleNLG is an NLG 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 is implemented as a java library and it requires java programming knowledge to be used. The following example shows the code needed to generate the sentences "My dog chases George." In this example a new phrase specification is created; the subject, verb, and complement are set; and finally, the document is realized and the sentence is displayed.

EXAMPLE 1

```
SPhraseSpec p = new SPhraseSpec();
p.setSubject(my dog);
p.setVerb(chase);
p.addComplement(George);
Realiser r = new Realiser();
System.out.println(r.realiseDocument(p));
```

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. The tasks performed when generating the sentence in example 1 were:

- the first letter of the sentence was capitalized,
- the verb was set in agreement with the subject (by default, third person singular is assumed),
- the words were put together in a grammatical form,
- whitespaces were inserted in the appropriate place between the words of the sentence, and
- a period was put at the end of the sentence.

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).While some of these features affect only the verb, others affect the structure of the whole sentence, as for example when it has to be expressed in the passive voice.

Because of the programming nature of SimpleNLG, it allows the user to define flexible templates by using programming variables in the sentence specification. As previously introduced, templates are frames that define sentences with some fixed parts and some variable parts. 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, however, 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.

3. Serious Games and the Need for NLG

The term serious games refer 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).

Feedback is an essential part of the learning process. Players need help not only navigating their way through the game environment, but also guidance that is designed to help them to reflect on their learning and understand the implications of their decisions, which ultimately leads to achievement of the desired learning objectives. Included in Gagné's events of learning (Gagné and Briggs, 1997) are three elements regarding guidance: providing learning guidance, providing feedback, and assessing performance. In serious games, we view this in-game guidance as a form of mentoring, and use it to help players incrementally improve their understanding, decision-making, and ultimately their performance.

As an example of a serious game, we briefly describe the game that we use for our experiments, 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. 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 every iteration of the game in order to keep the feedback consistent with the updated narrative. It is then necessary to include templates that will statically provide the basic information, combined with variable parts that adapt the narrative to the circumstances.

The template approach to textual information generation was first tried 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 is an example of a template used in the game ISO 14K.

EXAMPLE 2

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

In the previous example, PRONOUN_SUBJECTIVE would return either I or we depending on the ACTOR, PRONOUN_POSSESSIVE would return either my or our depending on the ACTOR, DEPARTMENT would take a value from a list of the company departments and representatives depending on the ACTION.

Using this template, 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. In our current work, we propose the use of a more flexible way of generating templates for the dialog of the games. We present a NLG Template Authoring Environment that 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.

4. 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 possible 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.

4.1 Design

A graphical design for the NLG Template Authoring Environment is shown in Figure 2. This also shows a simple example of a sentence with three variables and a dependency specified.

As shown in this figure, 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. For example the user could choose to add the past and future tenses to the default selected present tense. These verb options are presented to the user in a separate options window.

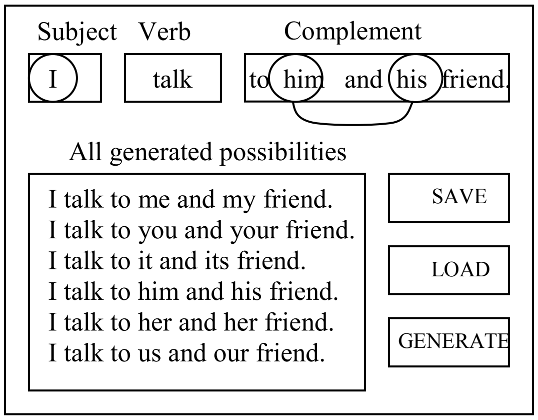

Subject
Verb
Complement
I
talk
to him and his friend.
All generated possibilities
I talk to me and my friend.
I talk to you and your friend.
I talk to it and its friend.
I talk to him and his friend.
I talk to her and her friend.
I talk to us and our friend.
SAVE
LOAD
GENERATE

Figure 2. NLG Template Authoring Environment

The system also 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 sentences, 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.

4.2 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 example sentence is parsed and stored in a structure that keeps separately the static parts and the variable parts (with information on where each variable part connects with the static part.)

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 purposes. Each variable knows how to update its current value with the next permitted (not-filtered) value of its type (i.e., a variable of type personal pronoun with current value "I" will update it to "you" when asked to get its next value by calling the method getNextValue, unless "you" has been specified as filtered out from the variables' values, in which case it will keep looking for the next permitted value). This is used by the method that generates all possible combinations of the variables' values[3]. This method visits the list of variables asking each of them to update their values. This method also checks the list of dependencies in order to filter out the pairs that do not satisfy them.

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.

As SimpleNLG requires the verb to be given as infinitive, a separate module is internally called before passing the verb to SimpleNLG. This module uses WordNet (Felbaum, 1998) in order to convert the main verb given in the sentence example to its infinitive form.

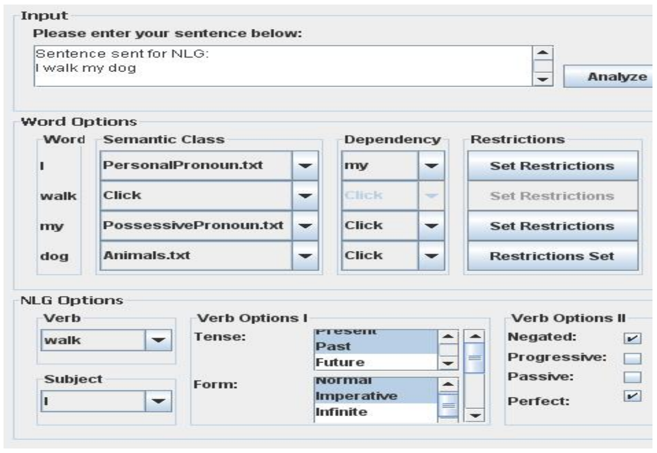
4.3 Interface

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 3.

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. The values of a semantic class are stored in a text file, which allows the user to create new semantic classes as needed. These files contain all the possible values and their respective syntactic information (person, number and gender) which will be used for agreement with the verb and for dependency between variables purposes. 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.

[3] The mechanism that generates all possible combinations of the variable values is currently implemented through a recursive method. It could in the future be replaced by a more effective method implemented using dynamic programming.

In the partial screenshot shown in Figure 3, 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).


Input

Please enter your sentence below:

Sentence sent for NLG:
I walk my dog

Word Options

Word	Semantic Class	Dependency	Restrictions
I	PersonalPronoun.txt	my	Set Restrictions
walk	Click	Click	Set Restrictions
my	PossessivePronoun.txt	Click	Set Restrictions
dog	Animals.txt	Click	Restrictions Set

NLG Options

Verb: walk

Subject: I

Verb Options I

Tense: [✓ Present] [✓ Past] Future

Form: [✓ Normal] [✓ Imperative] Infinite

Verb Options II

Negated: ✓

Progressive:

Passive:

Perfect: ✓

Figure 3. Graphical Interface

In Figure 3 we also see that 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). All these sentences will have the verb negated and will use the perfect tenses (as indicated by the extra verb options).

This interface also outputs a XML file containing the template information. This file is used for communication purposes between the graphical interface and the generation system. The generated templates in XML format are available for review and possible editing by experts. The modified templates can be directly provided to the generation system.[4]

We have trained two of our colleagues in the use of the system through this interface. 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. We are considering writing down a System’s Manual containing all the information provided during this training. This information could also be made available to the user through a help menu in the interface.

[4] This also allows computational linguistic experts to completely manually create templates for the sentences generation system.

5. Testing the System

In order to verify the correct functioning of the NLG Template Authoring Environment, we selected a set of sentence templates from the game. The templates were selected manually, while 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 and are identified in the rest of this section by their reference number or order in the table.

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).

Table 2. Testing examples

Ref. #	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.

The first template example has two variables of predefined closed class nouns, ACTORS and DEPARTMENTS. The latter is independent, while the former has a dependency with a variable of type personal pronoun in “object” form that could only take the values “me” or “us”. This template is used in the game when the actor/character available to help is the same actor/character that is providing the information. This template can be successfully generated with our system by declaring the variables, restricting the values of the pronoun variable, and establishing the dependency. When filtering non-valid sentences, the system will eliminate those cases where the value’s number of the variable ACTOR and the personal pronoun do not agree (i.e., it will only allow sentences that use “me” if the actor is singular, and sentences that use “us”, if the actor is plural). When creating this template, the user will have to be aware that the main verb is “to help” and indicate “could” as a modal to be used. This is important as otherwise SimpleNLG will modify the main verb in order to agree with the number of the subject. It is also necessary in case some of the options to change the main verb are specified.

Two examples of the generated sentences using the first template are shown below.

- The HR Training Manager (me) could help the Design Department.
- The Implementation Team (us) could help the Deputy Management Representative.

The second template is one that found a problem with our system and provided us with a reason and an opportunity to improve it. This example template also uses a variable of the closed class noun ACTOR together with the verb "to be" in the present tense, agreeing in number with the actor. It might seem trivial to indicate this dependency between the actor variable and the verb. But in our system the verbs are not treated as a regular variable (even when their values can be variable), but they are left for SimpleNLG to find the correct verb form. We needed then to inform SimpleNLG the number to which the verb should agree (by default it would assume singular). In this case we needed to inform SimpleNLG that the number to agree with would be the number of the variable ACTOR. We also have to consider the case when the subject number does not depend on a variable and is plural, as for example in a template where the subject is "The members of DEPARTMENT". To accommodate for these cases, we improved our system by asking the user to indicate whether the template's verb should agree with a variable value or it should be always used in plural or in singular.

The third template presents a dependency between a variable of type personal pronoun in the role of subject, and a variable of type possessive pronoun in the complement. Both variables accept only a pair of their possible values, and the dependency between them establishes that they have to agree in person and number. That is not a problem for our system. With respect to the verb, it is not used in its default tense (present) and therefore the user has to specify its infinitive and indicate the past tense as the only option.

In the fourth and fifth template, there is a personal pronoun variable taking the place of the subject, which should agree in person and number with the verb. This is, as mentioned before, left to SimpleNLG to solve. As the subject in these cases consists of only a personal pronoun and SimpleNLG can detect this fact, no extra information is required. In the fourth template, there is also a dependency between the personal pronoun variable in the subject role and the personal pronoun variable in the complement. Once again the person and number of these two variables have to agree, and the sentences not satisfying this restriction are filtered out by our system. Finally, for the fifth template the user is forced to specify that the verb "to be" has to be used in its past tense.

6. Comparison to 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.

6.1 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 using NLG.

Symbolic Authoring systems implemented using the WYSIWHM 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 by simply selecting options from pop up menus 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 WYSIWHM technique, the Symbolic Authoring systems are accessible to users who are not experts in either 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.

6.2 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 who are not aware of the systems' resources and constraints 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 of the process, options to complete the query are presented to the user in pop up 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 tracked 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 of a specific system.

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[5]. Even the values of the semantic classes used for the template variables are specified by the user.

7. 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 visual 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 our NLG Template Authoring Environment in a set of sentence templates from a digital-based interactive simulation game that covered different characteristics. When problems were discovered, we have improved the system to successfully produce the selected templates.

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 use of this interface will be evaluated in the context of the development of a new game. In order to further facilitate the feedback generating task we need to provide the users with the sentences identified and formatted according to the gaming system requirements. We will in the future include an option for the system to generate the final version of the feedbacks after the user has finished refining the templates and is satisfied with the sentences obtained.

We are currently 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 denoting 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.

Acknowledgements
This work is supported by the Ontario Centres of Excellence (OCE) and Precarn Incorporated.

References

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

[2] John A. Bateman. 2002. Natural Language Generation: an introduction and open-ended review of the state of the art.

[3] C. M. Christensen and M. E. Raynor. 2003. The Innovator's Solution: Creating and Sustaining Successful Growth. Harvard Business School Press.

[4] H. Dalianis. 1996. Concise Natural Language Generation from Formal Specifications, Ph.D. Thesis, (Teknologie Doktorsavhandling), 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.

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

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

[7] C. Felbaum. WordNet, an Electronic Lexical Database for English. Cambridge: MIT Press, 1998.

[8] R. M. Gagné and L. J. Briggs. 1997. Principles of instructional design (4th Ed.). Fort Worth, TX: Harcourt Brace-Jovanovich.

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

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

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

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

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

[5] Currently, the system does not perform a syntactic analysis of the created template. The different parts of the example sentence are given to SimpleNLG for generation. It is assumed that these parts are composed of English words of the expected syntactic type.