DEXTOR: Reduced Effort Authoring for Template-Based Natural Language Generation

Karthik S. Narayan[1], **Charles L. Isbell**[1], **David L. Roberts**[2]
Georgia Institute of Technology[1], North Carolina State University[2]
{karthik.narayan@, isbell@cc.}gatech.edu[1], robertsd@csc.ncsu.edu[2]

Abstract

A growing issue in the development of realistic and entertaining interactive games is the need for mechanisms that support ongoing natural language conversation between human players and artificial non-player characters. Unfortunately, many methods for automating and implementing natural language generation (NLG) induce a significant burden on the author, do not scale well, or require specialized linguistic knowledge. We formalize the notion of *typed-templates*, an extension of standard structures employed in template-based NLG. We further provide novel algorithms that, when applied to typed-templates, ameliorate the above issues by affording computational support for authoring and increased variation in utterance and scenario generation. We demonstrate the efficacy of typed-templates and the algorithms through a user study.

1 Introduction

As interactive entertainment and training experiences have grown in complexity and realism, there has been a growing need for robust technologies to shape and modify those experiences and interactions. One important kind of interaction is that of natural conversations between human players and artificial non-player characters. Natural language generation (NLG) is the field concerned with exactly this problem. Unfortunately, many of the methods developed in the NLG community impose a significant burden on authors to specify subject matter and, thus, do not scale well.

To address this issue, we introduce ***typed templates*** (t-templates) that are specifications of traditional templates that 1) allow for nesting, 2) restrict the nesting of templates, and 3) allow for several utterances per template. While not a revolutionary breakthrough in NLG template technology, t-templates can be combined with ***a novel compilation algorithm*** to enable significantly reduced authoring times for template-based NLG systems. We demonstrate that typed templates allow for improved authoring efficiency as well as varied utterance generation. Our main contributions are:

1. ***Typed-templates***, an extension of templates with recursive substitution whose advantages include reduced effort in language authoring and varied utterance generation.
2. ***An algorithm to compile*** t-templates that determines which t-templates are compatible with each other and produce grammatical, and varied, utterances, and a ***software implementation*** of that algorithm called DEXTOR.
3. ***Results from a user study*** demonstrating that t-templates and our compilation algorithms can significantly reduce language authoring time.

2 Other Approaches to NLG

Researchers have explored several methods of building NLG systems; modern systems largely employ canned text (Fum 1985; Isbell et al. 2001), features (Langkilde and Knight 1998; Walker, Rambow, and Rogati 2002; Barzilay and Lapata 2006), or templates (Becker 2002; Busemann and Horacek 1998; Mcroy, Channarukul, and Ali 2003). Consisting of purely hard-coded text, canned text is typically utilized for simple NLG purposes; when prompted for an utterance, the system returns one of the predefined statements. In feature-based approaches of NLG, authors specify various parts of speech and lexical items to the NLG system, which in turn synthesizes the information into natural language. In template-based approaches, authors use language structures call templates that produce utterances through a fill-in-the-blank mechanism, to realize text. For example, a template could have the utterance "[flight] is departing from [place] at [time]." where [flight], [place], and [time], are to be filled in with canned text or other templates.

There is some disagreement as to which method(s) are best. In domains where generalization is necessary (*e.g.*, tutoring systems and complex role playing games), using canned text does not scale well (Fox 1988; Theune 2003). While feature-based NLG systems can generalize to large domains, they are typically less efficient than other methods; further, effective usage often calls for designers to have specialized linguistic knowledge (Theune 2003). Template-based approaches have been adopted in many practical systems, but require authors to create template libraries, which becomes more challenging as systems scale (Channarukul, Mcroy, and Ali 2002; Reiter and Dale 2000).

We are primarily interested in template-based methods, as they are well used in practice and involve a shallow learning curve. Recently, researchers have introduced recursively substituted templates (filling in a blank may yield

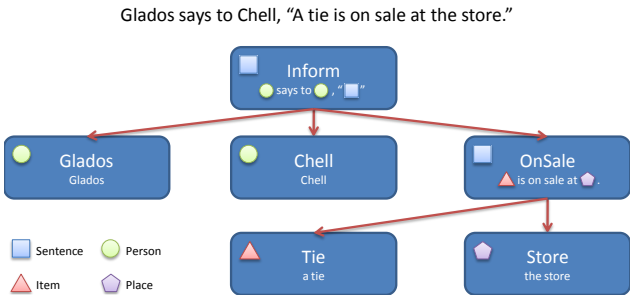

Figure 1: Example t-template utterance production. Shapes in the top-left of a box denote return types. Shapes within an utterance denote a sample parameter configuration.

more blanks), blurring what used to be a fine line between template and feature-based approaches (Piwek 2003; Deemter, Krahmer, and Theune 2005). This approach allows templates with similar structure to be more modular and reusable. We utilize typed templates, a derivation of recursively substituted templates that allow templates to have several utterances and restrict templates that can be substituted in blanks. We show that these additional specifications, which can be implemented on top of existing template-based systems, can yield significant improvements in varied natural language as well as reduced language authoring times.

3 Typed Templates

KSN - this seems really obvious from the description below.

T-templates are defined by four components: utterances, parameters, return types, and parameter configurations.

Utterances define text generated by templates. They may contain blanks that are in turn filled by utterances of other t-templates. A single t-template may have several utterances, one of which is chosen at random during generation.

Parameters are placeholders for gaps in utterances; utterances of bound t-templates fill in the values of the gaps. For clarity, we refer to bound t-templates as *subtemplates*.

Return types are author-defined strings that tag the semantic meaning of a template. A single template must contain at least one return type, but has no upper bound on the number of return types that it can have. Sample return types could be general (e.g. NOUN) or specialized (e.g. WEAPON); the only requirement is for all templates that form sentences to contain the return type, SENTENCE.

A ***parameter configuration***, or *PC*, is a mapping from t-template parameters to return types. A template can bind to a parameter only if the template contains the return type specified by the PC's corresponding return type. An ordered set of templates is said to be consistent with a PC if all templates can bind to their respective parameters. For a SENTENCE template T, a PC is considered *grammatical* if the utterances that result from binding subtemplates to the parameters of T form a grammatical sentence. Often t-templates may have several possible grammatical PCs; in general given a set of PCs for a t-template, when at least one of the PCs in the set is grammatical, the PC set is said to be *acceptable*. If T is a non-sentence template (*i.e.*, a phrase), then a PC for T is considered grammatical if T, along with it's parameters bound according to the PC, can be bound to a parameter of another template as part of a grammatical PC for that template. With this recursive definition, we state that all PCs for parameter free t-templates are grammatical.

Ultimately, the goal is to provide an acceptable PC set per template such that 1) the set's size is maximized and 2) each PC in the set is grammatical. The former allows for high template flexibility while the latter, as we observe later in the paper, reduces authoring time. In authoring, humans directly specify all components of a t-template excepting for an acceptable PC set; rather, the system computes an initial acceptable PC set in a compilation process, from which authors select appropriate grammatical PCs. The compilation process is discussed in detail in Section 4, and practical effects of compilation on reduced time is discussed in Section 7. To author a sentence, authors provide *dynamic text*, or a textual representation of a t-template tree, such as INFORM(GLADOS,CHELL,ONSALE(TIE, STORE)), to the NLG system. We believe that authoring dynamic text will be easier for authors because of the computational affordances we can leverage to aid in the authoring process.

As with traditional templates, t-templates are especially useful in cases where specialized text is necessary; t-templates are hand-crafted, so utterances are much more predictable than those generated by feature-based systems. While a computer can fill template blanks in expressing simple utterances, for more complex expressions, humans often need to recursively fill blanks while leaves could be filled by a computer. Often, game scenarios consist of a series of dialogue exchanges between a player and an NPC; as each sentence varies in expression, the full scenario produces utterances of significantly greater variance than what canned text allows for. Further, t-templates are simple in that they do not require specialized knowledge to use and have a shallow learning curve; in a study that we describe below, all subjects were able to learn and author a wide range of sentences using dynamic text within half an hour.

A key advantage that we emphasize is a suggestion feature that NLG systems using t-templates can implement to significantly reduce dynamic text authoring times. Once acceptable PC sets for each t-template have been determined, as an author begins typing dynamic text such as "INFORM(GLADOS,", the system can provide real-time suggestions regarding templates to fill in next. Potential subtemplates are suggested based on the acceptable PC set of the *root t-template*, the t-template at the root of the expressed template tree (in this case, INFORM). As the author starts binding subtemplates, the NLG system constructs the possible grammatical PCs of the head template that the author could possibly satisfy based on previously bound subtemplates (GLADOS). With this set of grammatical PCs, the system suggests all templates having the possible return types for the parameter that the author is trying to bind. We evaluate the effectiveness of this feature later in this paper.

The chief disadvantage of using t-templates lies in creating template libraries (not dynamic text authoring); ensuring

that templates are compatible with each other can often be time-consuming. As a result, as with templates, authoring large-scale t-template libraries may require non-trivial planning and organization. Authors can use the compilation process to ameliorate these problems by checking whether computed acceptable PC sets match expectations. Combining this method with existing methods to reduce authoring effort using conventional templates can significantly improve the maintainability of template-based NLG systems (Caropreso et al. 2009; Dannélls 2010).

4 Template Compilation

Manually constructing a complete acceptable PC set for each template in a large t-template library poses a large burden on authors. We use an algorithm that produces an acceptable PC set of a reduced size compared to the total number of possible PCs for a template (exponential in the number of parameters). Library designers then select grammatical PCs from the resulting sets. This process, referred to as *template compilation*, aids authors in constructing acceptable PC sets, as the reductions in the number of grammatical PCs are large (see Section 5). Template compilation is broken up into determining acceptable PC sets for SENTENCE templates and non-SENTENCE templates.

Compilation operates first on all SENTENCE t-templates and then proceeds to non-SENTENCE t-templates. As an initial pass, a *link grammar parser* (Lafferty, Sleator, and Temperley 1992) is used to determine if a PC is grammatical. After all SENTENCE t-templates are compiled using the link grammar parser, a library designer filters the resulting PCs because being grammatical is not a sufficient condition for an utterance to make sense. Subsequently, one by one each non-SENTENCE t-template attempts to compile by backchaining templates and PCs until a complete, grammatical, sentence is constructed. Again, a link grammar is used for a first pass at determining the acceptance of PCs, and a library designer filters the remaining. Often, a t-template may have to be marked to be learned in the future, particularly if no learned t-template can accept it. Once each non-SENTENCE template has attempted to compile, another sweep of non-SENTENCE compilation ensues unless: 1) all t-templates have successfully compiled or 2) no non-SENTENCE templates successfully compile on consecutive iterations. In the latter case, humans need to manually determine PCs for the remaining t-templates.

Advantages of compilation, reusability and reduced authoring time, arise due a reduced size of the set of PCs per template. With respect to reusability, compilation helps make designers aware of unseen possibilities in authoring with the constructed template library. Compilation reduces time in both t-template and dynamic text authoring by facilitating library construction; in order to determine whether newly constructed templates and return types integrate with each other properly, authors can use the compiler to check for intuitive grammatical PCs. Further, after compilation, we can construct the template suggestion tool described to reduce dynamic text authoring time. Intuitively, effort in both t-template and dynamic text authoring is reduced while reusability is boosted as the computed acceptable PC set more closely represents the set of PCs that only allow for grammatically correct utterances.

Table 1: Parameter configuration reductions for the sample template library. Numbers in parentheses indicate the percent reduction over the previous column.

Template	**Tot. PCs**	**PCs - comp.**	**PCs - filt.**
SENTENCE templates			
Eat(?, ?)	121	24 (80.16%)	6 (75%)
Kill(?, ?)	121	21 (82.64%)	5 (76.19%)
Visit(?, ?)	121	9 (92.56%)	5 (44.44%)
Walk(?, ?)	121	11 (90.91%)	1 (90.91%)
Inform(?, ?, ?)	1331	122 (90.83%)	3 (97.5%)
Pick(?, ?, ?)	1331	181 (86.40%)	3 (98.34%)
non-SENTENCE templates			
From(?)	11	7 (36.36%)	1 (85.71%)
Scary(?)	11	2 (81.82%)	1 (50.00%)
Thick(?)	11	4 (63.64%)	4 (0.00%)
Through(?)	11	8 (27.27%)	4 (50.00%)
Large(?)	11	5 (54.55%)	2 (60.00%)
Hungry(?)	11	2 (81.82%)	1 (50.00%)
And(?, ?)	121	66 (45.45%)	13 (80.30%)

5 Template Compilation Analysis

Our results from template compilation are encouraging: we saw an 80%-90% reduction in the number of PCs library designers needed to consider. We ran the template compilation process over a sample template library that consisted of 28 templates (12 atomic and 16 non-atomic) with 11 return types in total. For each template, 10 sample sentences were generated; if at least 9 of them produced acceptable sentences according to the link grammar, the corresponding PC was denoted as grammatical. We performed the final filtering process with the generated acceptable PC set. The results are presented in Table 1.

The goal of sentence compilation was to observe large reductions in the number of parameter configurations, as well as small reductions from manual filtering. Of the 6 SENTENCE t-templates, we generally noticed around 80-90% reductions in the number of PCs, leaving the rest for humans to filter. Note that a large reduction does not necessarily imply poorly constructed templates; for instance, in the VISIT template ("x visits y"), x must describe a person, and y must be a place or person. This drastically reduces the number of potential PCs, as x cannot be bound to a verb, adjective, *etc*., and y has similar restrictions. Examining human filtering on the simplified acceptable PC set, there were still several PCs that the link grammar incorrectly classified as semantically correct. Since non-SENTENCE reductions depend heavily on SENTENCE-reductions, it is important that human filtering takes place. Our results are positive, in that compilation reductions are high with low standard deviation (5.03%).

We present compilation time versus return type graphs for t-templates with two and three parameters in Figure 2. We do not vary the number of t-templates in the library, as compile

times per template only depend on the number of parameters in a template and the number of return types in the template library. Even with a fairly large number of return types, our algorithm can compile 2-templates (templates with 2 parameters) in reasonable times; however, compiling templates with more parameters takes exponentially longer. In a template library with 15 return types, the compiler takes 6.63 seconds to compile a single 3-template. In the same template library, the compiler takes 162.67 seconds to compile a single 4-template. This result is expected because the number of parameter configurations increases exponentially in the number of parameters. As this compilation process occurs offline and the size of domain-specific template libraries is generally moderate, exponential growth is not a concern.

6 Study Design

To evaluate if the suggestion feature reduces dynamic text authoring time, we implemented a "Dynamic tEXt generaTOR," called DEXTOR, that allows authors to create dynamic text using t-templates. In the following user study, as the system presents a series of statements, subjects author dynamic text that generates each statement. We hypothesize that DEXTOR's compilation will reduce the time spent constructing correct template bindings for dynamic text.

The study proceeded as follows:

- *Consent Form*. Subjects were presented with the consent form and were required to sign before continuing.
- *Familiarization*. We familiarize subjects with DEXTOR by having them author two sets of six statements using the client. Statements are simple sentences such as: Glados says to Chell, "There is a sale on ties.", and responses form dynamic text, such as: INFORM(GLADOS, CHELL, ONSALE(TIES)). In both the familiarization and testing question sets, we provide subjects with the root template.
- *Testing*. Subjects author another two sets of statements, which are timed. One set allows use of the suggestion feature while the other does not.
- *Exit survey*. Finally, subjects are presented with two questions to answer regarding the user study.

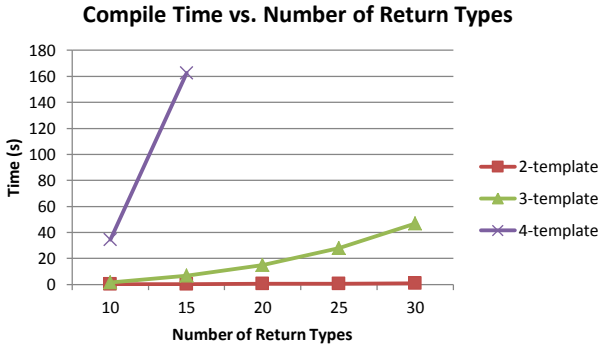

Figure 2: Compile times for t-templates with various parameter sizes in the LITTLEREDRIDINGHOOD template library.

All materials, including t-template libraries and statements to author were hand-crafted before the study. Human effort in determining PCs was primarily devoted towards filtering in compilation. We recruited 45 participants for the study from an introductory machine learning course taught at a local university. The students were either pursuing an undergraduate or graduate degree, and had at least three years of computer experience. Participation in the study was optional and no compensation was provided.

6.1 Familiarization, Testing, and Exit Survey

Familiarization is important in reducing learning curve effects that may impact authoring times in testing. Subjects encounter two sets of statements to author for practice before entering the testing stage. Both sets of questions that subjects author use a t-template library called GOLDILOCKS focusing on the children's story, "Goldilocks and the Three Bears." The first set of questions does not allow use of the suggestion feature, while the second set does.[1] Subjects are allowed to ask questions regarding the interface, as the familiarization portion of the study is not timed. Once subjects author the two sets of practice statements, they proceed to the timed statements. Again, each subject is given two sets of statements to author. Treatments for this study are varied by changing the template library behind the questions and whether suggestions are present. During the timed session, we leave the subject alone with the system to try to avoid observer-expectancy effects.

After authoring the timed statements, subjects are requested to participate in a brief two question exit survey. Participation is not obligatory. Responses are linked with testing question times, but are otherwise anonymous. The questions presented to the subjects are:

- The suggestion feature that was provided made statements easier to author. (Likert scale 1-5)
- Which of the following methods would you rather use in authoring sentences: templates (e.g. Inform(John, Bill, HelloWorld)) or just sentences (e.g. John says to Bill, "Hello world!")?

At the end of the two questions, there is a comment box for subjects to provide feedback. Once the subject clicks a submit button, all responses are stored, and an alert indicates the subject that he has completed the study.

6.2 Treatments

Treatments are applied during the testing component. Subjects experience two sets of statements to author, both of which draw from different t-template libraries, named HARRYPOTTER and LITTLEREDRIDINGHOOD. We constructed the libraries such that responses would have similar template tree structures, allowing for similar difficulties. We hypothesized their might be, but using a 3-way factorial ANOVA test failed to find any significant difference in question response times based on the story used ($p = 0.2092$). This was as we had hoped.

[1] By not presenting the suggestion feature, users effectively work with regular templates, and not t-templates.

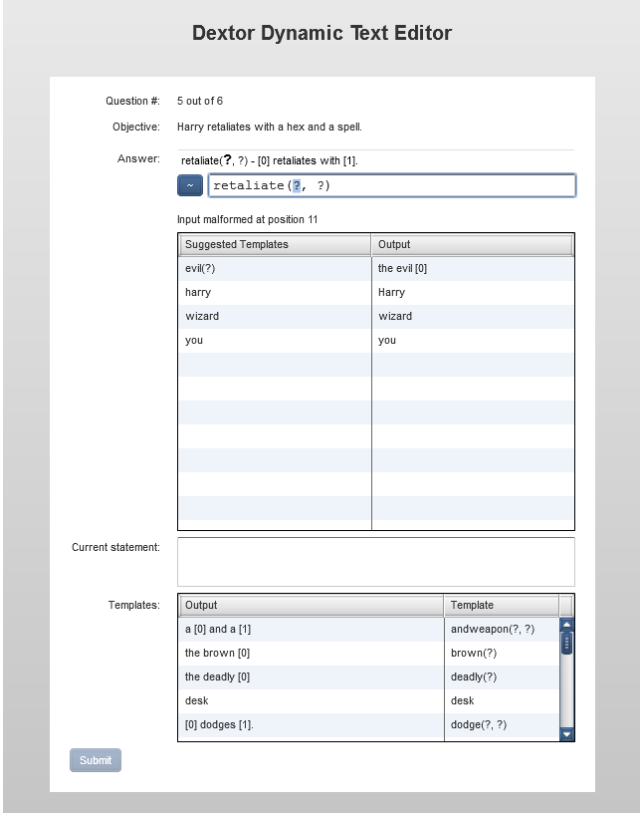

Figure 3: The user interface used for testing the suggestion feature (the large box in the middle). The same interface without the large box in the middle was used to test authoring without the suggestion feature.

Subjects are guaranteed to have one set of test statements with the suggestion feature and one without, enabling a longitudinal comparison. Varying the story and presence of suggestion feature aspects produce a total of four subject groups. Subjects were partitioned into these groups randomly. While we vary two factors in the study, we need to account for experience gained by subjects between the first and second set of questions. As such, we also include order as a factor to consider in the results. Of the 45 test subjects who participated in the study, 11 subjects were considered in each of the four groups; data for one subject was eliminated at random from the group with 12 subjects.

6.3 User Interface

In both practice and test sessions, users are initially greeted with a begin screen that indicates the name of the underlying template library, whether the suggestion tool is present, and the number of questions in the section. This begin screen also appears as subjects begin new question sets.

Subjects provided responses to questions using the interface in Figure 3. A target sentence is displayed, and subjects provide dynamic text to realize the target. All available templates are presented in a list, which we call the template box, towards the bottom of the screen. The template box is intentionally made where subjects have to scroll to find required templates, as language authors using large template libraries most likely need to do the same. Templates pertaining to the suggestion feature, if present, are in another list, the suggestion box, directly above the template box. Note that the template box is smaller than the suggestion box; we do this to encourage subjects to use the suggestion feature. The size difference does not affect the results, since at most five templates, the height of the template box, were ever listed at once in the suggestion box. Box sizes were identical across treatment groups, controlling for any interaction effects associated with the user interface and experiment group.

As seen in Figure 3, the initial answer box contains the root template, where each ? denotes a parameter. To produce dynamic text, subjects may either replace the question marks with template names using standard keyboard input, or double click the appropriate template in the template or suggestion boxes. Subjects proceed to the next statement once they provide valid dynamic text that generates the objective.

Table 2: A 3-way Factorial ANOVA test indicating effects of suggestions, the story, and order of appearance on cumulative times. Each row corresponds to a factor (e.g. "order") or to an interaction among factors (e.g. "order:story").

Variable	F-value	p-value
suggestion	82.5526	$\mathbf{4.007 \cdot 10^{-13}}$
order	0.2244	0.6373
story	1.6090	0.2092
suggestion:order	0.1509	0.6990
suggestion:story	0.6863	0.4105
order:story	0.0240	0.8774
suggestion:order:story	0.1800	0.6728

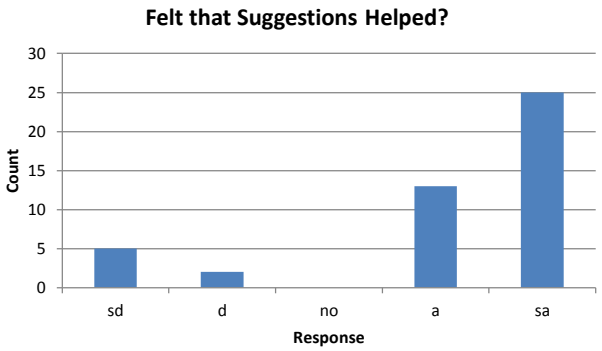

Figure 4: Distribution of responses corresponding to subjects' opinions on the usefulness of the suggestion feature.

7 Study Results

Our results fit our hypotheses with strong statistical significance; we can claim with confidence that the suggestion feature greatly reduces the burden on authoring. Table 2 summarizes results obtained from a three way factorial ANOVA

test. The factors considered were: 1) whether the suggestion tool was present, 2) the story presented, and 3) whether the question set was the first or second run for a subject. The suggestion feature accounts for a significant portion of the observed variance in time while the order and story do not appear to account for any of the observed variance.

7.1 Suggestion Feature Evaluation

The primary result of this study is that the suggestion feature reduces dynamic text authoring time. On average, subjects took around 2.1 times more time without the suggestion feature than with the suggestion feature: subjects took around 98.91 seconds ($\sigma = 35.44$ seconds) with suggestions and 207.42 seconds ($\sigma = 59.37$ seconds) without suggestions. This difference is highly significant with $p = 4.007 \cdot 10^{-13}$ (see "suggestion" variable in Table 1).

It is possible that the story or the order in which subjects received the suggestion feature could have influenced the times; however, the ANOVA tests do not indicate significant attributions to timing differences due to the story ($p = .2092$) nor ordering ($p = 0.6373$). Further, the results suggest that interactions between factors were low; from these observations, we believe that the suggestion feature was the primary contribution to reducing authoring time.

7.2 Exit Survey Results

Participants in each of the four categories seemed to rate the usefulness of the suggestion tool highly—of the 45 subjects tested, 38 (84%) either agreed or strongly agreed that the suggestion tool reduced efforts. Surprisingly, subjects were evenly matched between deciding whether language authoring was easier done with templates or sentences; 23 (51%) of the 45 subjects stated that they would rather use templates to author sentences while the remaining 22 stated that they would rather use sentence-based approaches.

8 Conclusion

T-templates offer several advantages over canned text and feature-based methods; as subjects in our user study demonstrated, authoring t-template libraries and dynamic text requires little to no prior linguistic or specialized knowledge. The shallow learning curve allows game designers to incorporate NLG components more easily, and with reduced authoring costs; our user study results indicate that exploiting t-template compatibilities, which requires a small cost due to compilation, results in less time required to author dynamic text with t-templates than with standard recursive templates. Two diverse lines of research that could improve the quality of DEXTOR as a whole include automated template generation and finer sentence discernability. Machine learning techniques could yield an algorithm that takes in a sentence and generates more varied sentences through template extraction. Through language smoothing techniques, one could potentially construct a better sentence classifier. Combining these methods, with compiled t-templates, natural language generation for games becomes more simple, and faster.

Acknowledgements

We would like to thank Aurel Lazar and William van Wagstaff for their useful insights. We would also like to thank the reviewers for their time in reviewing this paper.

References

Barzilay, R., and Lapata, M. 2006. Aggregation via set partitioning for natural language generation. HLT-NAACL '06, 359–366. Stroudsburg, PA, USA: Association for Computational Linguistics.

Becker, T. 2002. Practical, templatebased natural language generation with tag. In *Proceedings of TAG+6*.

Busemann, S., and Horacek, H. 1998. A flexible shallow approach to text generation. In *Proceedings of the Ninth International Workshop on Natural Language Generation*, 238–247.

Caropreso, M. F.; Inkpen, D.; Khan, S.; and Keshtkar, F. 2009. Novice-friendly natural language generation template authoring environment. In *Proceedings of the 22nd Canadian Conference on Artificial Intelligence: Advances in Artificial Intelligence*, Canadian AI '09, 195–198. Berlin, Heidelberg: Springer-Verlag.

Channarukul, S.; Mcroy, S. W.; and Ali, S. S. 2002. Jyag & idey: A template-based generator and its authoring tool. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics*.

Dannélls, D. 2010. Applying semantic frame theory to automate natural language template generation from ontology statements. INLG '10, 179–183. Stroudsburg, PA, USA: Association for Computational Linguistics.

Deemter, K. V.; Krahmer, E.; and Theune, M. 2005. Real versus template-based natural language generation: A false opposition? *Computational Linguistics* 31.

Fox, B. 1988. Robust learning environments: The issue of canned text. Technical report.

Fum, D. 1985. Natural language processing and the automatic acquisition of knowledge: a simulative approach. EACL '85, 79–88. Stroudsburg, PA, USA: Association for Computational Linguistics.

Isbell, C.; Shelton, C. R.; Kearns, M.; Singh, S.; and Stone, P. 2001. A social reinforcement learning agent. AGENTS '01, 377–384. New York, NY, USA: ACM.

Lafferty, J.; Sleator, D.; and Temperley, D. 1992. Grammatical trigrams: A probabilistic model of link grammar. Technical report, Pittsburgh, PA, USA.

Langkilde, I., and Knight, K. 1998. Generation that exploits corpus-based statistical knowledge. ACL '98, 704–710. Stroudsburg, PA, USA: Association for Computational Linguistics.

Mcroy, S. W.; Channarukul, S.; and Ali, S. S. 2003. An augmented template-based approach to text realization. *Nat. Lang. Eng.* 9:381–420.

Piwek, P. 2003. A flexible pragmatics-driven language generator for animated agents. EACL '03, 151–154. Stroudsburg, PA, USA: Association for Computational Linguistics.

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

Theune, M. A. 2003. Natural language generation for dialogue: system survey. Technical Report 2003-22.

Walker, M. A.; Rambow, O. C.; and Rogati, M. 2002. Training a sentence planner for spoken dialogue using boosting. *Computer Speech and Language*.