Toward Natural Language Generation by Humans

James Owen Ryan[1], **Andrew Max Fisher**[2], **Taylor Owen-Milner**[2],
Michael Mateas[1], **and Noah Wardrip-Fruin**[1]

[1] Expressive Intelligence Studio
[2] Department of Computer Science
University of California, Santa Cruz
{jor, michaelm, nwf}@soe.ucsc.edu
{anmfishe, towenmil}@ucsc.edu

Abstract

Natural language generation (NLG) has been featured in at most a handful of shipped games and interactive stories. This is certainly due to it being a very specialized practice, but another contributing factor is that the state of the art today, in terms of content quality, is simply inadequate. The major benefits of NLG are its alleviation of authorial burden and the capability it gives to a system of generating state-bespoke content, but we believe we can have these benefits without actually employing a full NLG pipeline. In this paper, we present the preliminary design of EXPRESSIONIST, an in-development mixed-initiative authoring tool that instantiates an authoring scheme residing somewhere between conventional NLG and conventional human content authoring. In this scheme, a human author plays the part of an NLG module in that she starts from a set of deep representations constructed for the game or story domain and proceeds to specify dialogic content that may express those representations. Rather than authoring static dialogue, the author defines a probabilistic context-free grammar that yields templated dialogue. This allows a human author to still harness a computer's generativity, but in a capacity in which it can be trusted: operating over probabilities and treelike control structures. Additional features of EXPRESSIONIST's design include arbitrary markup and realtime feedback showing currently valid derivations.

Introduction

Current dialogue-authoring practice, in which individuals or even teams of writers tirelessly produce huge amounts of content by hand, is largely seen as both untenable and constraining of the form of interactive narrative (Mateas 2007). Natural language generation (NLG) would seem to be the answer, and indeed there has been in the last decade a significant amount of work toward integrating NLG systems into playable media (Cavazza and Charles 2005; Rowe, Ha, and Lester 2008; Reed and others 2011; Walker and others 2013). But why then have we not seen shipped games or interactive stories that meaningfully incorporate NLG?

Outside of *Bot Colony* (Joseph 2012), we are not aware of any commercial game that has done this. Even in research contexts, we cannot name disseminated playable experiences that have featured significant NLG. The obvious reason for this is that NLG is an extremely difficult task. There is a huge amount of domain knowledge that its practitioners must have, and beyond this there remain several major technical challenges in incorporating NLG into playable media (Horswill 2014). We contend, however, that another major reason that NLG has yet to be popularly or even successfully incorporated into these media is that the current state of the art is simply inadequate.

Computer-generated dialogue, here and now, is so impoverished relative to the human-authored equivalent that its huge alleviation of authorial burden is not worth the attendant drop in content quality. This is not to say that NLG will never be successfully or popularly incorporated into games and interactive stories—we simply mean to express that the state of the art of NLG must be significantly advanced before this can happen. Moreover and more specifically, the state of the art of *expressive* NLG must markedly improve. We believe that major advances in NLG will not necessarily open the door to fully generative dialogue in expressive media, because the central concerns of NLG research are not the central concerns of expressive-NLG research (Mairesse and Walker 2011). The fundamental difference is that NLG seeks to generate *informative* text, while we seek to generate *expressive* text. This means that fully generative character dialogue may be much further off than has been anticipated.

So, if we cannot employ full NLG, what do we lose? First, we miss out on getting content essentially for free, which alleviates authorial burden, affords variability, and reduces repetitiveness. Beyond this obvious appeal, there is another significant advantage to using NLG in a game or interactive narrative. The typical NLG pipeline proceeds from a *deep representation* of an utterance to its realized surface form. This deep representation is in a machine-understandable form and specifies the semantic content of the utterance and potentially also its pragmatic force. If a game were to have an NLG module that could generate surface dialogue from arbitrary deep representations, it could employ character dialogue flexibly and expressively in a way that is tailored to system state. Of course, this would require the system to reason over its state in order to construct an appropriate deep

representation, but this is certainly feasible. While these advantages afforded by NLG would seem exclusive to the employment of that technique, we contend that we can still have them without actually generating dialogue from scratch.

In this paper, we present the preliminary design of EXPRESSIONIST, which is not an NLG module, but rather a *mixed-initiative authoring tool*. With this tool, a human author plays the part of an NLG module in that she starts from a set of deep representations constructed for the game or story domain and proceeds to specify dialogic content that may express those representations. This yields a database of lines of dialogue that are each explicitly annotated for the deep representations they express, allowing a game system to deploy dialogue in the way described above. Rather than authoring static dialogue, the author defines a probabilistic context-free grammar that yields templated dialogue. This allows dialogue components that have already been authored to be reused and makes authoring dialogue variations quite easy; moreover, the natural combinatorics of generative grammars yields a huge database of dialogue with relatively little authoring effort. By yielding *templated* dialogue, EXPRESSIONIST leaves certain kinds of variation that are highly sensitive to system state, such as the gender of a character, to be handled by the game system. Additionally, EXPRESSIONIST supports arbitrary markup and features realtime feedback that presents dialogue that may be generated given the production rules the author has already defined, ensuring that the tool's output will always be near the quality of conventionally authored dialogue. Due to space considerations, in this paper we assume a reader who is familiar with the basic terminology of generative grammars.

Prior Work

A number of systems have featured templated dialogue, such as *Curveship* (Montfort 2009), *Prom Week* (McCoy and others 2013), and *Versu* (Evans and Short 2014). This scheme allows an author to include variables in a line of dialogue that the system resolves when it gets displayed. Because variables may specify things like the person and gender of a pronominal reference to a character, or a command to randomly choose from a list of text segments, templated dialogue facilitates reuse across different contexts, with guarantees about correct conjugation and surface variation. By utilizing generative grammars, EXPRESSIONIST takes this templated approach to the extreme, with variables that may resolve to expressions that themselves have variables in them (and so forth recursively), all with realtime feedback showing surface derivations. Moreover, these surface derivations are actually templated dialogue in the vein of what these systems employ. In the sense that this dialogue is derived probabilistically using a Monte Carlo method, as we explain below, EXPRESSIONIST takes after the expressive-NLG system PERSONAGE (Mairesse and Walker 2011). In a sense, our system also follows a series of template-based NLG systems (Van Deemter, Krahmer, and Theune 2005), but a comparison to these is beyond the scope of this paper. As an endeavor in explicitly mapping underlying game states to natural language that may express those states, this project again follows *Curveship*, as well as our own work on procedurally recombining annotated *Prom Week* dialogue (Ryan and others 2014). Finally, EXPRESSIONIST is influenced by the grammar-based story-authoring tool Tracery (Compton, Filstrup, and Mateas 2014).

Tool Design

EXPRESSIONIST is a mixed-initiative tool for defining probabilistic context-free grammars that yield templated dialogue;[1] it is currently still in development. Our overarching design principle is to produce an authoring environment that maximally utilizes two complementary strengths of humans and computers—humans' deep knowledge of natural-language expressivity (and all its attendant nuances), and a computer's capacity to efficiently operate over probabilities and large treelike control structures—while simultaneously minimizing both entities' huge deficiencies in the converse. In other words, we want a tool that makes humans responsible for things they are good at and not for things they are bad at, and likewise with computers. The design of our tool's interface has four distinct panes—IN, TODO, WRITE, and OUT—which we will proceed to discuss in this section.

In the IN pane, an author populates a list of the *deep representations* that she will proceed from. EXPRESSIONIST does not enforce a specific notion of what a deep representation is, but conventionally (in NLG) it is a structured representation of the semantic content of an utterance, and possibly also its pragmatic force (*e.g.*, that it is a question). Less conventionally, but perhaps more powerfully, a deep representation could specify higher-level concerns—for example, what a line is intended to express to the player about its speaker (rather than its actual semantic content). In any event, a deep representation can be thought of as the logical form of a line of dialogue that would be uttered by a character in the game or story domain, one that specifies concerns that are primary to the expressive goals of the interactive experience. How the author actually formats her representations is up to her, but we envision something like predicate logic (*tell(speaker, interlocutor, location(item))*) being used. Whenever an author submits a new deep representation in the IN pane, it is added to a list of such representations in the TODO pane.

The core of EXPRESSIONIST is its WRITE pane, in which the author specifies production rules that will yield templated dialogic expressions of the deep representations in the TODO pane. To work on some representation, the author drags it from the TODO pane to the WRITE pane. Here, the author works in free text, but has at her disposal two syntactic constructs and an annotation field. The syntactic constructs are double brackets, which are used to specify *nonterminal symbols* (elements that will get expanded by a production rule), and single brackets, which are used to indicate *system variables*, by which we mean variables that must be resolved at runtime by the system deploying the dialogue. Let us consider an example production rule:

[[tell(speaker, interlocutor, location(item))]] → '[[greeting]], the [name(item)] is in the [location(item)].'

[1] We note that our tool could also be used to produce content that is not dialogue, for instance textual narration or in-game artifacts.

Here, `greeting` is a nonterminal symbol whose expansion must be specified by another production rule, and the single-bracketed expressions are system variables. The latter will eventually have to be filled in by a game system (not EXPRESSIONIST), which will bind `item` to some item in the game world and resolve `name(item)` and `location(item)` to strings of its name and current location respectively. It is in this sense that EXPRESSIONIST is a tool for authoring *templated* dialogue, like that used in *Prom Week* and *Versu*, rather than static dialogue. Because of this, an author must use a consistent formatting style for her tags for system variables (and for her deep representations) and then make her game system capable of parsing (and constructing) expressions in this format. Whenever the author specifies a new nonterminal symbol—*e.g.*, `greeting` in the example above—the system will add it to the TODO pane. The author can then drag such symbols back to the WRITE pane to specify how they may be expanded, just as she does for deep representations (which are treated like all other nonterminal symbols). Multiple production rules may be defined for any nonterminal symbol, and such a symbol (even a deep representation) may be reused on the right-hand side of production rules for any other nonterminal symbol.

Next, EXPRESSIONIST affords the specification of probability distributions for production rules. If a nonterminal symbol has multiple production rules, the author can express how often, relative to one another, each rule should be applied in expanding the symbol. As such, the central task in EXPRESSIONIST is to define a *probabilistic context-free grammar* (Jelinek, Lafferty, and Mercer 1992). For instance, if the nonterminal symbol `greeting` has rules specifying expansions to the strings 'Hello', 'Hi', and 'Hey', the author might attribute *application rates* `3`, `2`, and `1` to their respective rules, which would cause `greeting` to expand to 'Hello' three times more often than it would to 'Hey' and 1.5 times more often than it would to 'Hi'. Because nonterminal symbols may expand to templates with (arbitrarily many) nested nonterminal symbols, and because *these* have their own specified application rates, the full grammar that the author defines yields a rich probabilistic space of possible content derivations.

Further, any symbol can be annotated using arbitrary markup that is specified in a free-text field, and all derivations will inherit the markup of the nonterminal symbols that were expanded to yield them and the terminal symbols included in them. For instance, the speech act of a line of dialogue that is derived by the nonterminal symbol `greeting` might be attributed to that line by annotating the symbol itself with a tag `speech_act:greeting`. (Again, the author must do some work to make her game system capable of parsing the markup she attributes using the tool.) One possible type of markup could facilitate control structures in the game system. For instance, in a system with character personality modeling, a tag `extroversion:>5` could specify that a line of dialogue should only be uttered by characters whose personality component `extroversion` resolves (at runtime) to a value greater than `5`. Further, markup could be used to specify how lines might be sequenced into larger dialogic units, for example, by using tags like `sequence1:setup` and `sequence1:payoff`.

Our tool also features realtime feedback showing valid derivations of the production rules that an author has completed at some point. Specifically, if she completes a production rule for a nonterminal symbol and the expansion that it specifies includes no nonterminal symbols that cannot be expanded (given the other production rules she has defined), the author may ask the system to present valid derivations of that symbol. This allows the author to sanity-check the production rules that she has authored so far, to make sure that their derivations are grammatical and of sufficient quality.

Finally, in the OUT pane, the author asks the system to derive templated dialogue using the production rules she has specified and then requests that it export the resulting database in a structured format. Here, we have not decided which export format(s) we will support; one possibility would be to let the author specify her own format. For the actual derivation, we will likely implement something like a *Monte Carlo method* (Metropolis and Ulam 1949): for each deep representation, the system simulates its derivation many times, choosing expansions probabilistically according to the distributions specified by the author. Lastly, the OUT pane also affords exportation of a list of all deep representations and system variables that have valid derivations, given the grammar the author has defined. This is for convenience, since the author may take this list and work on her game system to make sure it can resolve all the system variables and potentially request dialogue for all the deep representations.

System Integration

After a session with EXPRESSIONIST, an author will have a database of templated lines of dialogue that are each annotated for their deep representations and for any other type of markup that the author has attributed. To use this dialogue, a game system will operate over the database in the following way. First, it must reason over its state to construct a deep representation for the line of dialogue that it wants to deploy. Next, it queries the database using that representation as a key and receives an array of all the templated lines of dialogue that are annotated as expressing it. Having this array, the system may then reason about the markup of its members to select a specific templated line. Finally, the game system fills in any variables in the template before displaying it.

A Brief Example

Let us consider the brief example of an author using EXPRESSIONIST to yield templated dialogue that expresses the deep representation *inquire_about_workplace(speaker, interlocutor, referent)*, which specifies dialogue by which a speaker may inquire with an interlocutor as to the workplace of some third character, the referent. After submitting this representation to the IN pane, the tool automatically adds it to the TODO pane; the author then drags it from there to the WRITE pane to begin working on it. At the WRITE pane, the author writes her first production rule:

[[inquire_about_workplace(speaker, interlocutor, referent)]] → '[[polite preliminary]][[ask where]] [name(referent)] [[work]]?'

After submitting this production rule, the system automatically adds the nonterminal symbols `polite preliminary`, `ask where`, and `work` to the TODO pane (because they are double-bracketed). Next, the author proceeds to drag `ask where` to the WRITE pane and composes production rules by which it may expand to 'Where does', 'Where exactly does', and 'Where the hell does', respectively, and attributes application rates to each rule. Additionally, she annotates the symbol `ask where` with the tag `speech_act:request` and the terminal symbol 'Where the hell does' with the tag `agreeableness:low`. From here, she moves on to `polite preliminary` and writes two rules for it: one expands to the empty string, and the other to 'I'm wondering [[modal request]] help me: ', which she annotates with the tag `agreeableness:high` and whose specification causes `modal request` to be added to the TODO pane. Subsequently, she brings `modal request` to the WRITE pane and specifies rules that expand it to 'if you could', 'if you would', and 'whether you could'. Again, she assigns application rates to each rule. All that remains in the TODO pane now is the nonterminal symbol `work`; the author specifies rules that expand the symbol to 'work', 'work at', and 'earn [poss_pron(referent)] keep', before assigning application rates to each rule.

Having done this, the system indicates that surface derivations can now be yielded for the deep representation, because all of its component nonterminal symbols have production rules specified for them (and so forth recursively). The user now asks the system for an example derivation. Starting from the production rule for the deep representation, the system probabilistically expands (using the application rates attributed to the production rules) its nonterminal symbols from left to right to give the example derivation 'I'm wondering if you could help me: Where does [name(referent)] work?'. Other valid derivations would include 'Where does [name(referent)] work at?' and 'Where the hell does [name(referent)] earn [poss_pron(referent)] keep?'. In fact, this simple grammar yields a total of 36 templates that each express the deep representation differently.

Finally, upon moving to the OUT pane, the author requests to export her dialogue, which causes the system to enact its Monte Carlo derivation procedure. The result is a database of dialogue templates that are each associated with a probability, as well as the annotations `deep_rep:inquire_about_workplace(speaker, interlocutor, referent)` and `speech_act:request` (since all of them inherit the mark-up of the symbol `ask where`); additionally, some will have the tags `agreeableness:high` and `agreeableness:low` (inherited from the symbols 'I'm wondering [[modal request]] help me: ' and 'Where the hell does' respectively). To actually deploy the dialogue, the author's game system will have to retrieve the 36 variants from the database (by using the deep representation as a key), reason over their respective annotations to choose the most appropriate variant, bind the variable `referent` to the appropriate character, and fill in the template gaps `name(referent)` (and `poss_pron(referent)`, if that is included) with that character's name (and gendered possessive pronoun).

Conclusion

In this paper, we have outlined the preliminary design of EXPRESSIONIST, a mixed-initiative authoring tool in which a human plays the part of an NLG module. We have argued that this yields much of the the benefit of having such a system, but without its attendant reduction in the quality of generated content. Using our tool, a human author starts from a set of deep representations constructed for the game or story domain and proceeds to specify dialogic content that may express those representations. Rather than authoring static dialogue, the author defines a probabilistic context-free grammar that yields templated dialogue. This allows a human author to still harness a computer's generativity, but in a capacity in which it can be trusted: operating over probabilities and treelike control structures. EXPRESSIONIST is currently in development.

References

Cavazza, M., and Charles, F. 2005. Dialogue generation in character-based interactive storytelling. In *Proc. AIIDE*.

Compton, K.; Filstrup, B.; and Mateas, M. 2014. Tracery: Approachable story grammar authoring for casual users. In *Proc. (int)7*.

Evans, R., and Short, E. 2014. Versu—a simulationist storytelling system. *Computational Intelligence and AI in Games*.

Horswill, I. D. 2014. Architectural issues for compositional dialog in games. In *Proc. GAMNLP*.

Jelinek, F.; Lafferty, J. D.; and Mercer, R. L. 1992. *Basic methods of probabilistic context free grammars*.

Joseph, E. 2012. Bot colony—a video game featuring intelligent language-based interaction with the characters. In *Proc. GAMNLP*.

Mairesse, F., and Walker, M. A. 2011. Controlling user perceptions of linguistic style. *Computational Linguistics*.

Mateas, M. 2007. The authoring bottleneck in creating ai-based interactive stories [panel]. In *Proc. INT*.

McCoy, J., et al. 2013. Prom Week: Designing past the game/story dilemma. In *Proc. FDG*.

Metropolis, N., and Ulam, S. 1949. The Monte Carlo method. *Journal of the American Statistical Association*.

Montfort, N. 2009. Curveship: An interactive fiction system for interactive narrating. In *Proc. CALC*.

Reed, A. A., et al. 2011. A step towards the future of role-playing games: The SpyFeet mobile RPG project. In *Proc. AIIDE*.

Rowe, J. P.; Ha, E. Y.; and Lester, J. C. 2008. Archetype-driven character dialogue generation for interactive narrative. In *Proc. IVA*.

Ryan, J. O., et al. 2014. Combinatorial dialogue authoring. In *Proc. ICIDS*.

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

Walker, M. A., et al. 2013. Using expressive language generation to increase authorial leverage. In *Proc. INT6*.