# Evaluating Natural Language Descriptions Generated in a Workspace-Based Architecture

**George A. Wright**[1]
[1]Cognitive Science Research Group
School of Electronic Engineering and Computer Science
Queen Mary University of London
george.a.wright@qmul.ac.uk

**Matthew Purver**[1,2]
[2]Department of Knowledge Technologies
Jožef Stefan Institute
Ljubljana, Slovenia
m.purver@qmul.ac.uk

## Abstract

This paper concerns the evaluation of a workspace architecture for generating natural language descriptions, including methods for evaluating both its output and its own self-evaluation. Herein are details of preliminary results from evaluation of an early iteration of the architecture operating in the domain of weather. The domain is not typically seen as creative, but provides a simple testbed for the architecture and evaluation methodology. The program does not yet match humans in terms of fluency of language, factual correctness, and how completely the input is described, but human judges did find the program's output easier to read than human generated texts. Planned improvements to the program also described in the paper will incorporate self-monitoring and better self-evaluation with the aim of producing descriptions that are more fluently written and more accurate.

## Introduction

This paper describes work towards a self-evaluating architecture for language generation first described in (Wright and Purver 2020) and a method for evaluating the architecture by comparing human judgements of its output with its own self-evaluation. This iteration of the architecture operates in a toy domain: making simple descriptions of temperatures on a static, two-dimensional map but serves as an initial framework on which future versions performing more ambitious tasks can be built.

## Theoretical Background

According to Fauconnier (1994), linguistic meaning is organized in mental spaces and according to Fauconnier and Turner (2002), creativity involves the projection of structures across mental spaces, often with the help of frames. Such processes cannot involve a deterministic search for an optimum, but instead a constant competition between structures evolving in a bubble chamber of mental spaces, only some of which become available to consciousness (Fauconnier and Turner 2002, p.321).

The architecture described below implements the projection of structures across spaces while making use of an enzymes-in-cytoplasm metaphor of cognition similar to that proposed by Barrett (2005) which allows for a chaotic interaction of processes in a shared workspace or *bubble chamber*. These include processes of language production and comprehension which also interact when humans use language (Pickering and Garrod 2013). In this architecture, self-comprehension and self-evaluation are important because they help to determine which of the competing intermediate structures are used in future processing. An overall *satisfaction* score also affects how randomly processes occur. Evaluation of this architecture therefore takes into account not only the finished outputs of the program, but also its method for self-evaluation.

## The Planned Architecture

The architecture has a *bubble chamber* and a *coderack*. The bubble chamber contains a network of concepts, frames, and their instantiations spread across a number of conceptual and working spaces. These are the long- and short-term memory of the program. The best, most useful structures *bubble* to the top of the program's attention as their activation increases.

The coderack, borrowed from Copycat (1993) and related work (Hofstadter and FARG 1995) contains a collection of codelets, (small tasks to be carried out), each of which has an urgency influencing the likelihood it runs. Codelets correspond to the enzymes of Barrett's metaphor. They are selected from the coderack with a degree of randomness determined by the program's *satisfaction*, a score of the quality of active structures in the bubble chamber (a structure's quality is determined by evaluation codelets). High satisfaction leads to less random codelet selection thus more deterministic processing whereas low satisfaction leads to more randomness and opens a broader set of pathways to be explored. Self-evaluation is central to the architecture and is therefore important to consider when judging its performance.

Most codelets make a small change to the bubble chamber, for example by building a new node or link, or by changing a structure's activation. All structures, including representations of the input, parse trees, and output text are built incrementally in this manner. Codelets also change the coderack by adding a follow-up codelet. Some codelets operate exclusively on the coderack by adding or removing codelets in order to ensure that the coderack does not become empty or overcrowded.

This style of architecture shares similarities with models based on Baars' (1997) Global Workspace Theory such as (Misztal and Indurkhya 2014) which has *experts* performing tasks in a shared workspace. But, where as codelets in this architecture are restricted to performing small operations, some experts in Miszatal and Indurkhya's architecture such as the *metaphor expert* operate at a much higher level and perform tasks comparable in complexity to work performed by a large collective of codelets.

## Engagement and Reflection Cycles

According to Sharples (1998), the creative writing process involves a cyclic alternation between engagement (producing new ideas) and reflection (evaluating work so far). This has been implemented in models of language generation (Pérez y Pérez and Sharples 2001) as well as other models of creativity (Pérez y Pérez, de Cossío, and Guerrero 2013). The E-R model is a relatively high-level view of cognition which does not recognize the more intertwined nature of production and comprehension described by Pickering and Garrod (2013).

This architecture contains something like an engagement-reflection cycle but at multiple levels of abstraction and, due to the stochasticity of the coderack, with less rigidity.

**Codelet Cycles** Most codelets operating in the bubble chamber belong to one of four types: *suggesters*, *builders*, *evaluators*, and *selectors*.

Suggesters find an element in the input such as a temperature on a map and suggest a possible structure that can be built for that element. For example, a temperature could be labeled as HOT or in the SOUTH, two temperatures could be combined into a single chunk if they are similar, two temperatures could be connected with a MORE or LESS relation, or a SAMENESS correspondence could be recognized between a chunk in the input and an item in a frame.

Having performed a classification, a suggester codelet places a builder codelet on the coderack with an urgency matching its confidence in its suggestion. If the builder codelet is run, the relevant structure is built and the builder codelet then places an evaluator codelet on the coderack.

Evaluator codelets determine the quality of the structure according to the same classifier as the suggester. Since certain classifications can be context dependent, for example a part-of-speech label may depend on how a word is used in a sentence, the classification of a structure by the time the evaluator is run may differ from when the structure was first suggested. The evaluator assigns a quality score to the structure and then places a selector codelet on the coderack.

Selector codelets compare two competing structures, for example two incompatible labels, and boost the activation of one while depressing the activation of the other such that only one structure is likely to be used in further processing. Higher quality structures are more likely to receive a boost in activation. Selector codelets also place another suggester on the coderack thus completing a cycle at the fine-grained level of workspace structures.

If a codelet fizzles because the bubble chamber does not contain the right conditions or if a follow-up has low urgency and never runs, the cycle breaks. Meanwhile new cycles are created as *factory* codelets add new suggesters and evaluators to the coderack so that processing does not stop prematurely.

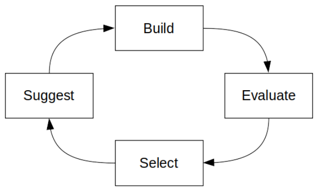

Figure 1: The lowest-level "cycle of engagement and reflection" at the level of individual nodes and links in the bubble chamber.

**View Cycles** The architecture implements the *simplex networks* of Fauconnier and Turner (2002, p.120-2), which connect elements in an input space to elements in a frame and then elements in both the input and the frame to new elements in an ouptut space. Since this is a language generating program, the frames are templates with slots to be filled in according to the input. The output is a text which describes the original input using the template structure. Each network exists within a *view* based upon the *Worldview* of the Tabletop model of analogy-making (French 1995). All structures within a view must be consistent with one another.

The architecture also uses views for self-monitoring. *Monitoring views* contain an output text, a semantic parse of the text and a set of correspondences between elements of the parse and the original input. The purpose of a monitoring view is to check that a text both makes sense and is an accurate description of at least part of the original input.

Texts which have been matched to part of the original input are made available for further processing inside higher level simplex networks using discourse frames. This allows for a recursion of simplex networks as described by Fauconnier and Turner (2002, p.151) and produces a cycle of engagement and reflection at the higher level of fragments of text which emerges from the cycles of engagement and reflection at the lower level of individual nodes and links.

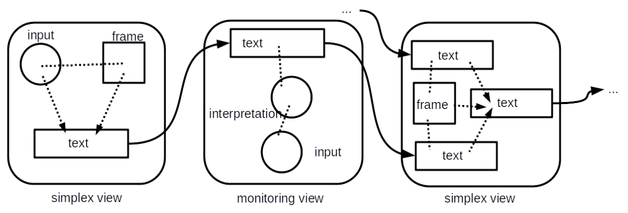

Figure 2: A higher level "cycle of engagement and reflection" at the level of pieces of text.

## The Current Implementation

The implementation of the architecture described above operates within a simple domain, describing temperatures on a map. This requires a small knowledge base and allows for focus to be placed on the mechanisms of the architecture.

Implementation is still in an early stage and lacks much of the self-monitoring provided by monitoring views.

In order to get the program to output text, a *publisher* codelet occasionally runs, which finds templates that have had their slots filled and outputs the resulting text. Current outputs are therefore short and lack discourse structure, but the evaluation of these outputs provides a base-line upon which future iterations of the model can improve.

The current implementation's satisfaction is calculated as the mean of the product of each bubble chamber structure's quality and activation. This means the satisfaction is higher when the most active structures have a high quality and lower when active structures have a low quality or high quality structures have a low activation. But, as discussed below, this results in a satisfaction score which fails to take into account a more global perspective on the bubble chamber.

## Evaluating The Program

The relatively transparent nature of the program allows it to be evaluated in a number of ways: the intermediate representations it builds when processing the input, its textual output, its understanding of its own textual output (through syntactic and semantic parses), and its satisfaction score for its output can all be seen and evaluated by external observers.

Below is described a subjective and intrinsic evaluation of outputs of the system implemented thus far - a survey which evaluated the system in isolation from any practical application and according to human value judgements. Such surveys commonly focus on two main criteria: the quality of a text, and its accuracy relative to the input (Gatt and Krahmer 2018, p.124).

### The Survey

Human subjects in the survey were asked to compare two of the program's outputs for each input. They had to answer four questions for each pair:

1. Which text is easier to understand?
2. Which text is more fluent?
3. Which text is more factually correct?
4. Which text represents the map more completely?

Respondents could answer each question in one of three ways: the first text is better than the second, the second text is better than the first, or the two texts are approximately equal.

The aim of the first two questions was to capture the linguistic quality of the texts, while the aim of the final two questions was to capture their accuracy as descriptions of the input map. Survey respondents only saw the map after the first two questions so that any inaccuracies in the description would not influence the quality score.

Human subjects had to compare two outputs rather than score them on a scale as it is unclear what the criteria are for high or low scores, especially when viewing the first few outputs from a program. Furthermore, Belz and Kow (2010) compared preference-based evaluation to score-based evaluation and found that preference-based evaluation results in less variance between respondents.

Since the computer program provides its own satisfaction score for its work, human evaluation can also be used to check if its internal measure of satisfaction matches with human judgements or if its method for calculating satisfaction could be improved. Since the program only has a single number to describe its "satisfaction", there is no one-to-one correspondence with the questions used to judge linguistic quality and factual accuracy. The score is also an absolute number rather than a preference judgement. Nevertheless, rankings based on human judgements and rankings based on the program's internal score ought roughly to align.

Methods for evaluating the creativity of computer programs commonly try to rate the novelty of outputs as well as their quality, see for example (Ritchie 2007). This is not attempted here since the domain is so simple and the outputs are so short that no output is likely to be in any way novel. It is hopefully clear though, that this architecture could in theory be applied to a more complex domain that would allow for more exciting outputs where novelty would be worth considering.

### Generation of Texts for the Survey

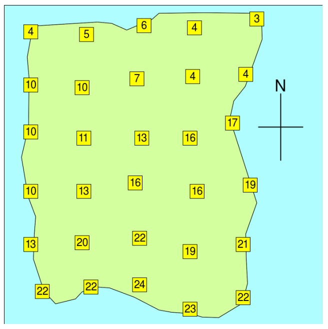

Figure 3: The first input as displayed to survey respondents. Numbers show temperatures in centigrade.

The survey was carried out using four different inputs to the program. For each input, the program was run ten times, and three outputs were randomly selected. Outputs all took the form of simple statements of fact. Added to these outputs were two human-generated descriptions which were gathered from a separate survey. For each input, one description was selected which was written with detailed, full sentences while the second description was brief and often written in note-form. At no point were the respondents told that they were evaluating machine-generated or human-generated text. The texts used for the first input were:

A (Human) "The temperature is cold in the north but progressively warm moving south, reaching 24 degrees."

B (Computer) "It is hot in the southeast."

C (Computer) "It is mild in the northeast."

D (Computer) "The north is mild."

E (Human) "Cool in the north, warm in the south."

The purpose of including human-generated outputs was to check that respondents (on Amazon Mechanical Turk)

| | Easiness | Fluency | Correctness | Completeness |
|---|---|---|---|---|
| 1 | B | A* | E* | E* |
| 2 | D | D | A* | A* |
| 3 | A* | E* | B | B |
| 4 | C | C | D | D |
| 5 | E* | B | C | C |

Table 1: Average rankings according to the pairwise preferences of survey respondents for texts describing the first input. *Human-generated texts.

understood the task and were not pressing random buttons. A respondent who understands and pays attention to the task ought at this point broadly to prefer the human-generated texts. In future, improved iterations of the program ought to surpass the briefer note-like human-generated texts. Outputs of future iterations can also be compared to outputs of the current iteration to check if changes to the architecture result in improved results.

## Results of the Survey

The results of the survey are unsurprising in that they show that the program is overall below human-level performance, but they also highlight certain issues that should be taken into account in future evaluation.

It should first be noted that respondents of the survey did not show a high degree of agreement. The Fleiss' Kappa scores were 0.342 for ease of understanding, 0.238 for fluency, 0.484 for factual correctness, and 0.485 for completeness (to calculate Fleiss' Kappa the three possible answers to each question were treated as a category). This may in part be due to the fact that respondents had a different understanding of the questions they were being asked: future surveys should make more clear what each of these terms means, especially *correctness* and *completeness* which some respondents seemed to treat as the same. Low agreement may also have been caused by arbitrary decisions being made when similar computer outputs were compared. The survey also only had 7 respondents. In future, surveys using more respondents may result in better agreement.

Respondents on average, ranked human-generated texts above computer-generated texts along the dimensions of fluency, correctness, and completeness. But they found computer-generated texts easier to understand. A similar result was found by Reiter *et al* (2005, p.138) who found that readers preferred a computer program's weather forecasts to those written by human's due to greater consistency in the program's word choices. It is likely to be the case that more rigid and precise computer programs will always outperform humans along this dimension within small data-to-text applications, but this should be less easy to achieve in more complex domains requiring narrative or explanation. Achieving greater ease-of-understanding scores will therefore not be a priority in future work on this architecture where the aim is to achieve something closer to human-like creativity in language generation.

For the most part, no preference was shown for one text's easiness or fluency over another when two computer-generated outputs were displayed side-by-side. This is understandable given that computer-generated outputs all followed one of two sentence patterns: `the [location] is [temperature]` and `it is [temperature] in the [location]`. Some computer-generated texts used words which did not match well with the input map and were therefore not preferred when it came to correctness and completeness.

There may have been some confounding variables which affected respondents' evaluation of the text, for example the length of the sentences being compared. Future evaluation should consider the extent to which such variables influence people's preferences.

## Evaluating the Program's Self-Evaluation

The linguistic similarity of the outputs is reflected in the computer program's satisfaction scores. The 40 runs executed for the purpose of evaluation had a mean satisfaction score of 0.704 with a standard deviation of 0.065. But, the program even had similar satisfaction scores in the 12 cases when it failed to produce an output before timing out after 30,000 codelets were run. This is because the satisfaction score is based entirely on the quality and activation of individual, low-level structures in the bubble chamber and does not take into account more global criteria for satisfaction such as the proportion of the input that has been described. It is clear that an improved metric for the satisfaction of the program is required but unfortunately it is difficult to compare different metrics when the program consistently produces similar outputs.

## Future Work

There are many improvements that can be made to the architecture, most urgent of which is the implementation of *monitoring views* in which codelets will build correspondences between the semantic parse of a text and the original input in order to check whether or not the text is factually correct and also to measure the extent to which the input has been described. This should reduce the incidence of inaccurate outputs.

The addition of discourse frames which the program can use to combine phrases and produce longer sentences should result in more fluent and complete descriptions of the input.

Furthermore, changes in higher level structures such as greater coverage of the input and improved discourse structure must be reflected in the program's satisfaction score. Future rounds of evaluation can consider alternative methods for calculating satisfaction and compare human rankings with the program's scoring of its own output.

## Conclusion

This paper has provided the outline of a planned architecture for language generation and a method for evaluating the architecture by eliciting human judgements of its output and comparing those judgements to the program's internal self-evaluation. Described in the paper is an early iteration of the architecture which lacks some of the core components required for self-monitoring and more complex discourse structuring. The program's outputs are therefore still disappointing, but outputs of future versions of the program can be compared with its current outputs to see the extent to which greater self-monitoring improves performance.

## Acknowledgments

Purver is partially supported by the EPSRC under grant EP/S033564/1, and by the European Union's Horizon 2020 program under grant agreement 825153 (EMBEDDIA, Cross-Lingual Embeddings for Less- Represented Languages in European News Media). The results of this publication reflect only the authors' views and the Commission is not responsible for any use that may be made of the information it contains.

## References

Baars, B. J. 1997. *In the Theater of Consciousness: The Workspace of the Mind*. Oxford University Press.

Barrett, H. C. 2005. Enzymatic computation and cognitive modularity. *Mind & Language* 20(3):259–287.

Belz, A., and Kow, E. 2010. Comparing rating scales and preference judgements in language evaluation. In *Proceedings of the 6th International Conference on Natural Language Generation*, 7–15.

Fauconnier, G., and Turner, M. 2002. *The Way We Think: Conceptual Blending and the Mind's Hidden Complexities*. Basic Books.

Fauconnier, G. 1994. *Mental Spaces: Aspects of Meaning Construction in Natural Langugae*. Cambridge University Press.

French, R. M. 1995. *The Subtlety of Sameness: A Theory and Computer Model of Analogy-Making*. The MIT Press.

Gatt, A., and Krahmer, E. 2018. Survey of the state of the art in natural language generation: Core tasks, applications, and evaluation. *Journal of Artificial Intelligence Research* 61:65–170.

Hofstadter, D., and FARG. 1995. *Fluid Concepts and Creative Analogies*. Basic Books.

Misztal, J., and Indurkhya, B. 2014. Poetry generation system with an emotional personality. In *Proceddings of the Fifth International Conference on Computational Creativity*, 72–81.

Mitchell, M. 1993. *Analogy-Making as Perception: A Computer Model*. The MIT Press.

Pickering, M. J., and Garrod, S. 2013. An integrated theory of language production and comprehension. *Behavioural and Brain Sciences* 36:329–392.

Pérez y Pérez, R., and Sharples, M. 2001. MEXICA: A computer model of a cognitive account of creative writing. *Journal of Experimental and Theoretical Artificial Intelligence* 13.

Pérez y Pérez, R.; de Cossío, M. G.; and Guerrero, I. 2013. A computer model for the generation of visual compositions. In *Proceedings of the Fourth International Conference on Computational Creativity*, 105–112.

Reiter, E.; Sripada, S.; Hunter, J.; Yu, J.; and Davy, I. 2005. Choosing words in computer-generated weather forecasts. *Artificial Intelligence* 167(1-2):137–169.

Ritchie, G. 2007. Some empirical criteria for attributing creativity to a computer program. *Minds & Machines* 17:67–99.

Sharples, M. 1998. *How We Write: Writing as Creative Design*. Routledge.

Wright, G., and Purver, M. 2020. Creative language generation in a society of engagement and reflection. In *Proccedings of the 11th International Conference on Computational Creativity*, 169–172.