

# Concurrent Constraint Programming as a Whiteboard Architecture for Probabilistic NLG

Irene Langkilde-Geary  
Computer Science Department  
Brigham Young University  
Provo, UT irenelg@cs.byu.edu

## 1 Abstract

In a recent attempt to generalize across the architectures of 19 representative generator systems, Cahill et al. proposed a whiteboard as a reference design [Cahill *et al.*, 1999; 2000]. They argued that a whiteboard was general enough to subsume the variety of architectures surveyed and could serve as a standard to facilitate the sharing and reuse of processing resources. A whiteboard architecture divides processing into knowledge-centric functional modules and models communication between modules with a single repository of data. The data stored in a whiteboard is added cumulatively by extending or copying existing data, but not changing or removing anything. A whiteboard thus generalizes further what had previously been considered the most general architecture for generation, namely a blackboard architecture, in which changes to the data repository are destructive, not cumulative.

Existing large-scale practical systems to-date have actually used other architectures (typically a pipeline) for the sake of simplicity. However, one ultimate goal in generation is to design a general-purpose system that is not only large-scale, but that works equally well across a variety of applications from dialog to machine translation to automatic summarization. A whiteboard architecture is necessary to achieve such generality, since different applications will provide somewhat different kinds of information as input, and will thus need to perform subtasks in different orders.

Historically there has been much discussion and puzzlement as to what sort of representation ought to be used as the input to a generator. For example, there has been uncertainty about what level of abstraction would be appropriate—deep (semantic) or shallow (syntactic). Deep inputs allow clients such as dialog systems to delegate much more work to a generator. However, deep (or even shallow) representations may be difficult for client applications like machine translation or summarization to construct.

Furthermore, the literature has noted that many real-world applications would find it convenient to generate output using a mix of canned text and templates in addition to fine-grained representations. It would therefore be desirable for a general-purpose generator to support all three forms of granularity.

Finally, the degree to which an input may be underspecified can vary. In general, client applications are likely to prefer having to provide the littlest amount of information necessary to get reasonable output, because that makes using the

generator much easier. Sometimes the information needed to specify the input is simply unavailable. For example, in machine translation from Japanese to English, the generator needs to know the number feature for nouns, but that is not typically explicit in Japanese. The generator must then make its best guess, or choose a default. On the other hand, there are times when a client may want precise control over some or all of the output generated, for example to maintain coherency across multiple sentences or utterances.

Can a single system possibly do all this at once? I suggest that it can by allowing inputs to vary along the three dimensions of abstraction level, granularity, and degree of specification independently. Using a declarative (not transformational) representation of natural language sentences, this task can be modeled as a constraint satisfaction/optimization problem where each word is associated with a set of features, and some subset of the features have values given in the input while the remainder must be solved for. The framework is neutral with respect to which features are known initially versus unknown. The task then is to solve for the unknown values given the known ones using a set of constraints on the values of the features. The constraints may include rules and/or probabilistic dependencies or any other declarative specification.

One of the main features in this model is the identity of each word's syntactic head, and another is the position of a word in the syntax tree with respect to its head and siblings (assuming a dependency-style representation). A third feature is the linear position in the final sequence of words. Assuming projectivity, the tree position and parent ID features are sufficient to determine the linear order of words of a realized sentence given an input. Conversely, there is no reason why one cannot solve for the parent ID and tree positions of each word in a sentence given the linear positions.

Thus, such an approach to generation could also do parsing. Handling inputs with different granularities then is simply a matter of parsing canned text or template fragments and integrating the resulting fine-grained specifications with other fine-grained specifications to produce an output.

The recent maturation of concurrent constraint programming (CCP) makes such a formulation possible. Constraint programming is a flexible, general, and principled framework for efficiently solving hard combinatorial problems. It integrates techniques developed in AI and Operations Research

and embeds them within a general-purpose programming language. Concurrent constraint programs are the most powerful form of constraint programs, because they increase the modularity and declarativeness with which a program can be formulated. In particular, they offer a greater degree of order-independence than non-concurrent formulations. A concurrent constraint program's behavior is implicitly analogous to that of a whiteboard architecture (although the specific representation and processing details may differ from those proposed by [Cahill *et al.*, 2002]).

In this talk I will present my lab's work on formulating probabilistic parsing/generation as a CCP. This early stage work helps to demonstrate the feasibility and effectiveness of the approach.

## References

- [Cahill *et al.*, 1999] L. Cahill, C. Doran, R. Evans, C. Mellish, D. Paiva, M. Reape, D. Scott, and N. Tipper. In search of a reference architecture for NLG systems. In *Proc. EWNLG*, 1999.
- [Cahill *et al.*, 2000] L. Cahill, C. Doran, R. Evans, R. Kibble, C. Mellish, D. Paiva, M. Reape, D. Scott, and N. Tipper. Enabling resource sharing in language generation: an abstract reference architecture. In *Proc. LREC*, 2000.
- [Cahill *et al.*, 2002] L. Cahill, R. Evans, C. Mellish, D. Paiva, M. Reape, and D. Scott. The RAGS reference manual. Technical report, Univ. of Brighton and Univ. of Edinburgh, 2002.