



Concept String White Paper

Andrew N. Edmonds PhD

Scientio, LLC.

Saturday, 26 December 2009

Version 13

<http://www.scientio.com>

support@scientio.com

Contents

Diagrams 1

Introduction 2

Background 2

The derivation of Concept Strings 3

Implementation..... 4

Comparing Concept Strings 5

Storing Concept Strings 7

Modularity and Language 8

Extensibility 9

Classes and API..... 9

Usage and applications..... 11

Diagrams

Figure 1, Concept string structure 4

Figure 2, Concept String comparison..... 6

Figure 3, Suffix tree structure 8

Figure 4, Concept string processing block diagram..... 9

Figure 5, Concept string classes..... 10

Introduction

Concept strings are an invention of Scientio LLC, first mooted in 2007 and elaborated in various ways after that date. They are a means for representing text that encapsulates not only the text itself, but parts of speech information, (POS), and the various concepts that might be meant for each word. Concept strings encapsulate the set of meanings that can be inferred from the enclosed text.

A concept string is a structure, in computer science terms, that contains an array of words, corresponding to the sequence of words in the source text, each annotated with its inferred part of speech and an array of possible meanings.

The usefulness of this technology is based on the kinds of processing that can be applied to these concept strings that cannot be applied to text on its own. As well as being able to convert text to a concept string representation and back again, we can efficiently compare, index, store and retrieve concept strings in large volumes and at high speeds.

As well as textual and part of speech comparison we can also compare concepts, i.e. the possible meaning of texts, and it is this capacity that gives Concept Strings their unique power.

Background

There is a large volume of literature on the subject of text mining, but two threads can be followed over the past few years. They could be loosely labelled the statistical and the natural language processing (NLP) threads.

Both threads are concerned with words rather than characters. There are approximately 40,000 words in the English language. In both approaches the text mining tools will attempt to relate a given word in a stream of new text to one of those 40,000 or flag the text as a proper name, a spelling mistake or both.

Text processing based on statistics will usually attempt to stem words. Stemming is the process of finding the root part of a word. So for instance “writing”, ”written”, ”wrote” are all examples of the verb “write” and a stemmer will replace them with the root word.

NLP, on the other hand, needs to conserve the information lost in stemming, and will instead use a POS tagger, a software tool that uses probabilistic models of English grammar to infer the parts of speech, tense and voice of each word.

Having stemmed text, a statistical processing engine will then typically ignore word order and create a vector of word frequencies from that text. This approach is frequently called “bag of words”, in that word order is ignored for processing convenience.

These word frequency vectors are mathematically and computationally tractable and can be used in a variety of ways. They are the raw material of modern search engines, such as Google and Bing. It is possible to use them to classify documents by training learning engines using techniques such as Naive Bayes and Support Vector Machines.

NLP solutions have not prospered as well, because words annotated with parts of speech are not tractable: words have no natural structure other than the grammatical. The field of Computational Linguistics has made use of such structures to investigate real world grammatical usage, but otherwise application development has been slow.

A WordNet is a model of a language, combining the functions of a dictionary and a thesaurus in a computerised form. The raw material of a WordNet is a *synset*, a collection of synonyms that share the same meaning, with the associated part of speech for that collection. Each synset has a unique index, and most importantly for our application, has a variety of pointers to other synsets. The kinds of pointers vary according to the part of speech, but typical pointers point to antonyms, holonyms, meronyms, hypernyms and hyponyms.

By following these pointers it is possible to build a fully connected object oriented model of the WordNet's language. The key elements are the trees created by meronymy and hypernymy.

Meronymy describes "is a part of" relationships, while hypernymy describes "is a kind of" relationships. All nouns concepts are either child concepts of another, or parent concepts of others, or both, by one or both of these relationship types.

Hypernymy trees are typically very large, and the English language has only 11 of them for nouns, i.e. all noun concepts are either child or root nodes of one of 11 trees.

The derivation of Concept Strings

In a previous work the author described a technique based on hypernymy trees, for creating signatures from documents that proved to be very useful in comparing and indexing documents in large corpora.

The paper showed that a distance measure could be created between two documents that modelled well the human perception of document difference. This technique converts words to concepts, using a lookup into the WordNet model, but ignores word order. It might be termed a "bag of concepts" technique.

This technique was not well suited, however, to relating documents by conceptual content. I.e. it could not effectively be used to retrieve documents that were about the same kinds of things from a large database.

The reasons for this failure were perceived to be the coarseness of the 11 dimension signatures, and the inaccuracy of the word to concept lookup process. In fairness, the problem itself is nebulous and ill formed: what a document is "about" is a subjective analysis, and different readers may form totally different opinions about a given document. Also, any one document is frequently about many things. For instance, although this document is about concept strings, different parts of it are about technology, other kinds of text mining and various other topics. However, comparing documents by conceptual content is one of those tasks where there might be debate about what is a correct retrieval, but there is no doubt about what an incorrect one looks like.

We can be reasonably sure that the smaller the samples we take from a document, the more likely the samples are to be about only one thing. The smallest practical unit that we can hope to compare would be a sentence, and so our Concept String technology encapsulates text at this scale.

There are two sources of ambiguity in converting words to concepts: The fact that a given word can relate to multiple parts of speech, and the fact that each of those relationships may exist with multiple concepts. So for example “read” may be a noun as in “that was a good read” or a verb “I read War and Peace”. Many words still have multiple meanings, even if you have decided on the part of speech meant in a given usage.

Both of these ambiguities can only be reduced by considering the context around a given word. A POS parser can be used to determine, with reasonable, but not absolute accuracy, the part of speech. Inferring which concept is meant once part of speech is decided is more complicated, but the solution most definitely depends on context.

It is for these reasons we began to look into performing document analysis and comparison at the sentence level, and hence created Concept Strings.

Implementation

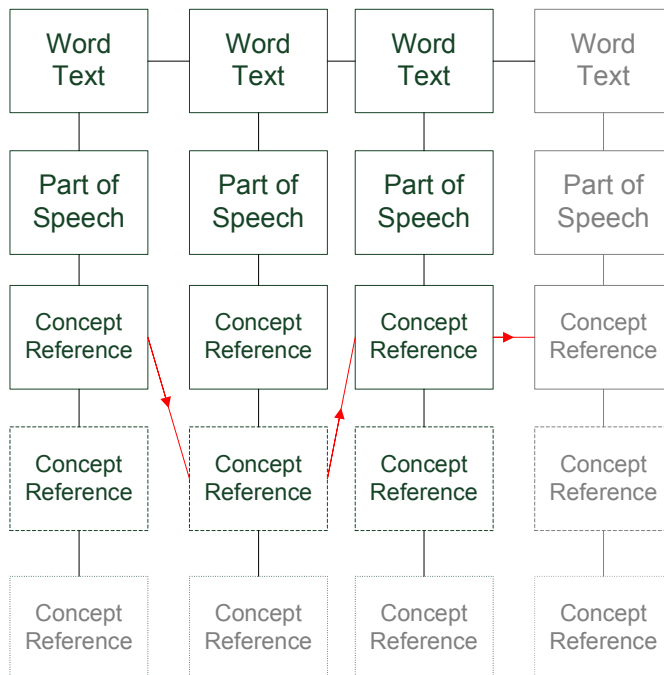


Figure 1, Concept string structure

Each possible path through the concept references (example in red) represents an alternate meaning for the string

Comparing Concept Strings

If we have two concept strings, how can we compare them in a way that is meaningful for the kinds of applications we envisage? Concept strings consist of a sequence of words, tagged with POS and possible concepts. We can first of all ensure that the sequences of POS tags match over some part of the sequences, i.e. that the same sub-sequence occurs in both. Once we've determined this we can then compare the matching sub sequences, word by word.

Obviously if the words are textually the same, we have a match, but we want to consider the concepts too. Each concept resides somewhere in a hypernymy relationship with other concepts. In order to detect similarity we must search through the neighbourhood of each tree to see if one concept is the parent of the other or if they have a near mutual ancestor. We must do this for each pair of concepts from each word pair in the matching sequence.

Hypernymy encapsulates the 'is a kind of' relationship, so we are entirely justified in searching the trees in this way, and we can expect matches found to, in one sense at least of their possible meanings, match well in meaning.

Using this process we can create a similarity metric that can be used to measure the distance between these two concept strings.

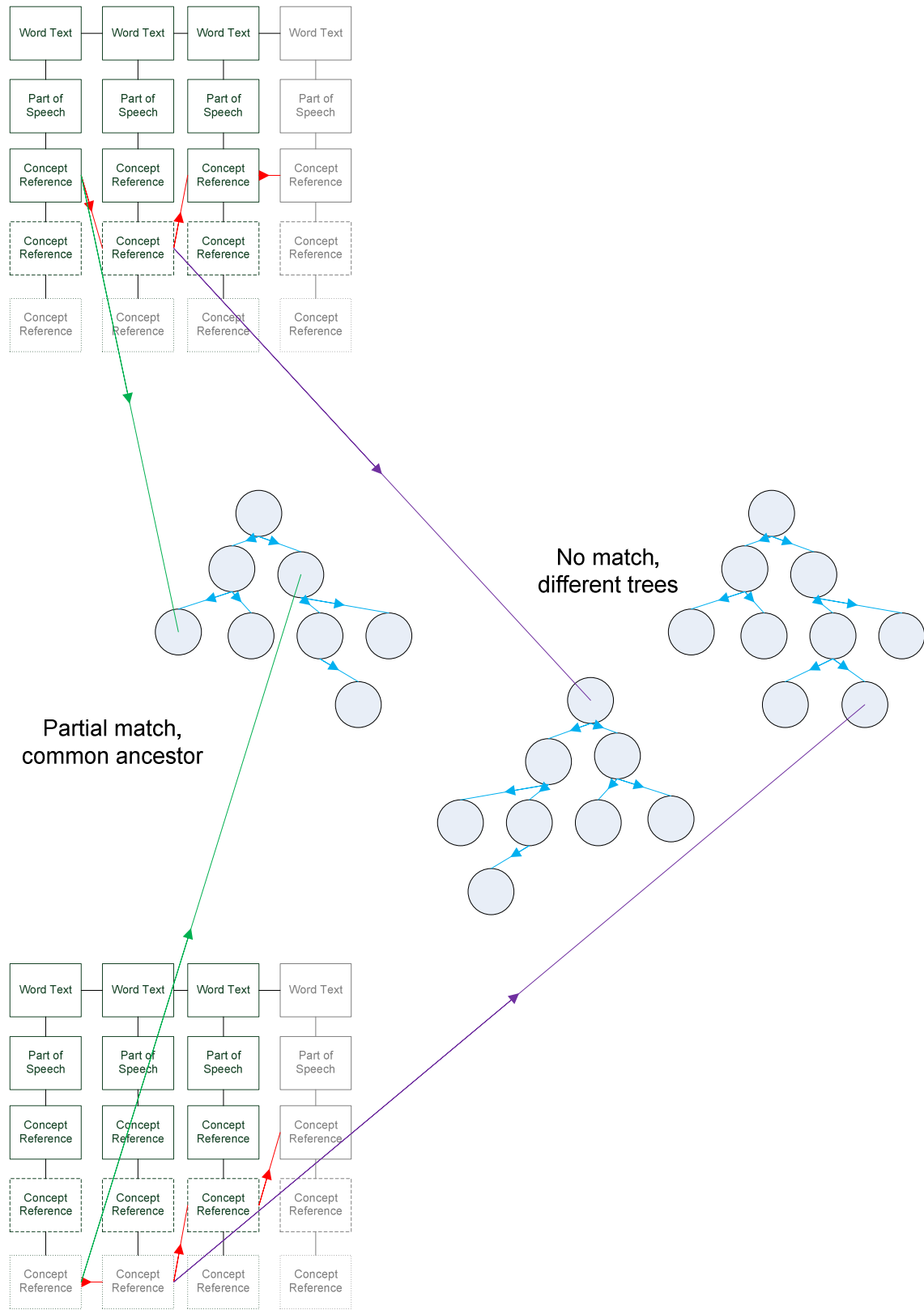


Figure 2, Concept String comparison

It is clear that the above processing is potentially very compute-intensive. Scientio has identified several methods to accelerate and linearize this processing, the details of which are outside the scope of this document.

Storing Concept Strings

In a typical application we will be trying to match a new concept string with a collection of existing strings. One can easily think of several brute force ways to do this, but a key consideration must be that computation and storage requirements are well behaved as we increase the size of the collection. This severely limits the choice.

A similar problem occurs in the domain of gene sequencing in Biotech, and the solution taken there is to use Suffix Trees.

Suffix trees have exceedingly good characteristics in this regard. They can be used to match incoming sequences in linear time, and their space requirements are nearly linear for increasing database size. The generalized form of a suffix tree can store multiple sequences, and incoming sequences can be matched with any stored sequence or sub-sequence. The result of a matching process is a list of matched sequences, the offsets for each at which the match occurs, the length of each match, and the offset in the incoming sequence at which the match occurs.

Scientio has created an adapted and extended suffix tree storage system that stores and retrieves concept strings with the above characteristics. Again much proprietary design and optimisation was required to create this architecture, the details of which are outside the scope of this document.

Currently this store is memory-resident, but can be copied to and from file storage, and to extend the size of the store beyond memory constraints an object-oriented database can be used.

Each Concept String stored in the database can be tagged with an arbitrary object and retains the source document index and document offset.

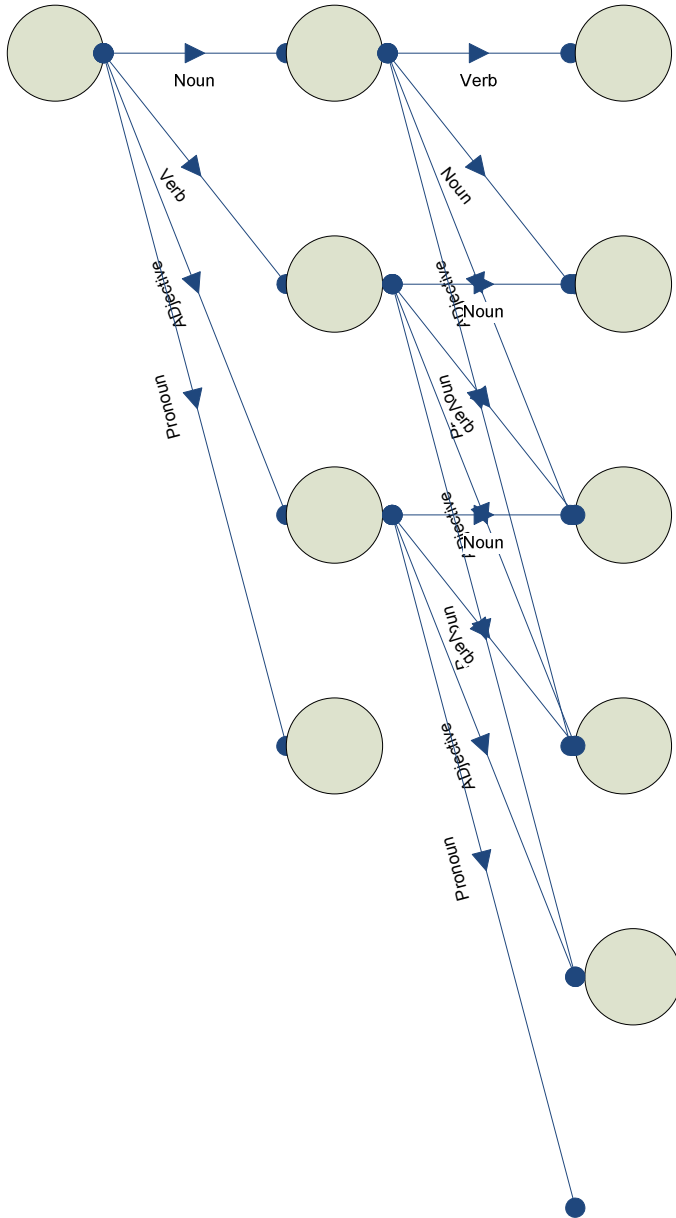


Figure 3, Suffix tree structure

Modularity and Language

The concept string generating classes have been constructed so that various key items can be interchanged. The focus of Concept strings is currently the recognition of text that has the same meaning as some stored text, not the machine understanding of text itself. In this light we need not concern ourselves too deeply in linguistic arguments about the proper set of POS tags recognised by a given tagger, or indeed the fact that WordNets created for different languages are not always structurally consistent. The important thing is that both elements work on large volumes of text, and that we do not try to compare concept strings created by different tagger, WorldNet combinations. Also, note that while POS taggers are considered to be only around

90% accurate, as long as the errors are consistent we can still expect better accuracy from Concept Strings. Currently Scientio have worked in depth with Princeton's US English WordNet and the EU's Spanish WordNet.

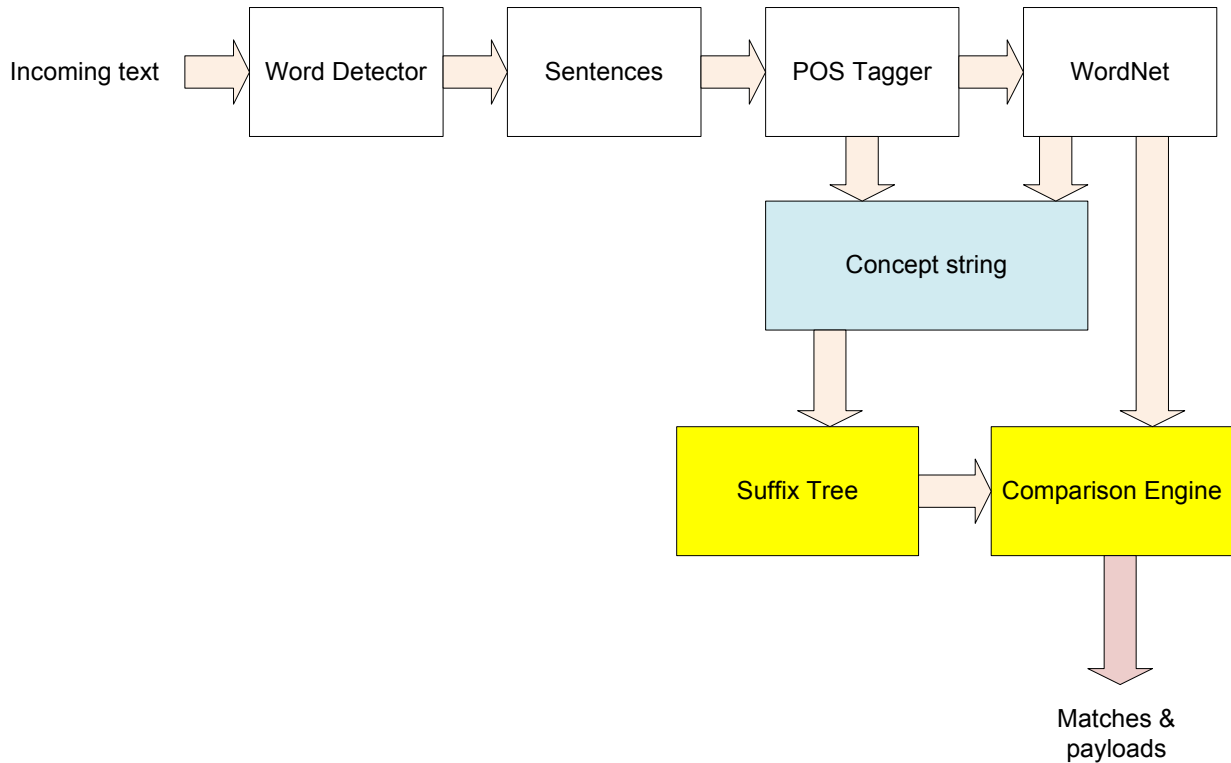


Figure 4, Concept string processing block diagram

Extensibility

The Concept String architecture has been designed to be extensible. Two key areas that are under development are improving concept disambiguation through mining associations in representative corpora, and the retention of context across strings.

Clearly language is an evolving medium, and WordNets are somewhat fixed in time. The internal WordNet based model created for concept strings is arranged to be extensible. New usages can be added, specialized vocabulary may be specified and alternate spellings and support for slang or text language may be easily incorporated.

Classes and API

The two classes below encapsulate the externally accessible classes of the Concept String Library.

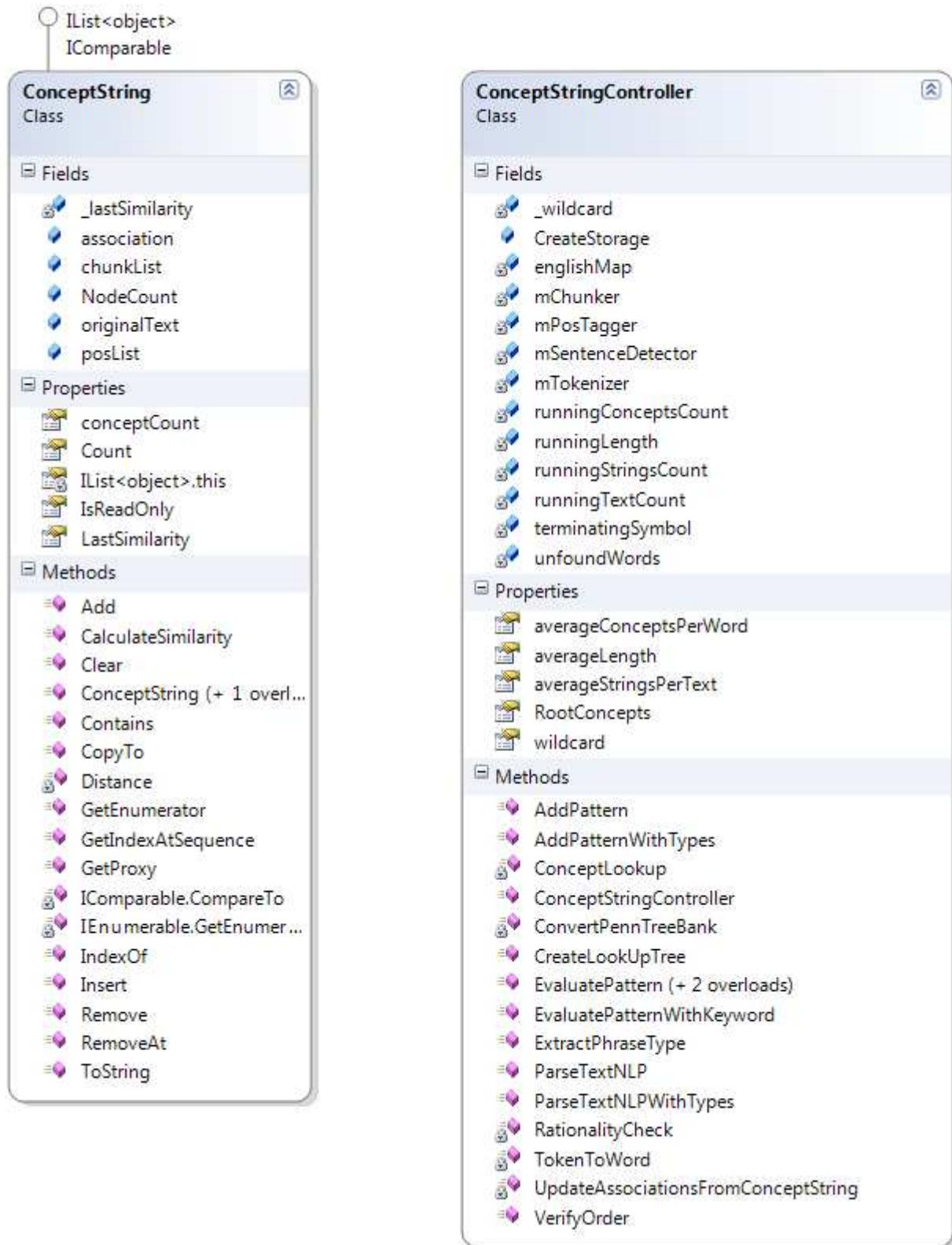


Figure 5, Concept string classes

Usage and applications

Applications for concept strings are manifold. They offer the ability for the user to create simple template text that can then be matched against other text in a filtering process. The matches are based on concepts, rather than words, and so we can expect to match phrases and sentences with the same or similar meaning. While the user needs to supply template text, which may just be previously detected examples of the kind of text that they wish to filter, they do not need to consider every combination of words that might convey a particular meaning; rather they need to consider far fewer template texts that explore variations in grammar. Thus new recognition schemes can be set up rapidly with minimal effort.

The direct applications of this technique include homeland security, detecting unauthorized disclosures or comments in outgoing emails, detecting cyber bullying and abuse in emails, games and chat rooms, ‘Smart’ user interfaces such as Chat bots and many more.

Since the memory demands of this technique are light, we can even imagine the ‘full text search’ of existing databases being replaced by ‘full concept search’!