Natural Language Generation From Ontologies

Van Duc Nguyen
Computer Science Department
[New Mexico State University, USA]
vnguyen@cs.nmsu.edu

Abstract

The paper addresses the problem of automatic generation of natural language descriptions for ontology-described artifacts. The motivation for the work is the challenge of providing textual descriptions of automatically generated scientific workflows (e.g., paragraphs that scientists can include in their publications). The extended abstract presents a system which generates descriptions of sets of atoms derived from a collection of ontologies. The system, called `nlgPhylogeny`, demonstrates the feasibility of the task in the *Phylotastic* project, that aims at providing evolutionary biologists with a platform for automatic generation of phylogenetic trees given some suitable inputs. `nlgPhylogeny` utilizes the fact that the Grammatical Framework (GF) is suitable for the natural language generation (NLG) task; the abstract shows how elements of the ontologies in Phylotastic, such as web services, inputs and outputs of web services, can be encoded in GF for the NLG task.

2012 ACM Subject Classification Computing methodologies → Logic programming and answer set programming, Information systems → World Wide Web → Web services, Computing methodologies → Artificial intelligence → Natural language processing → Natural language generation

Keywords and phrases Phylotastic, Grammatical Framework

Digital Object Identifier 10.4230/OASIcs.ICLP.2018.22

1 Introduction

In many applications whose users are not proficient in computer programming, it is of the utmost important to be able to communicate the results of a computation to the users in an easily understandable way (e.g., text rather than a complex data structure). The problem of generating natural language explanations has been explored in several research efforts. For example, the problem has been studied in the context of question-answering systems,[1] providing recommendations,[2], etc. With the proliferation of spoken dialogue systems and conversational agents on mobile robots, phones, etc., verbal interfaces such as Amazon Echo and Google Home for human-robot-interaction, and the availability of text-to-speech programs such as the TTSReader Extension,[3] the application arena of systems capable of generating natural language representation will just become larger.

In this paper, we describe a system called `nlgPhylogeny`for generating natural language descriptions of collections of atoms derived from a set of ontologies. The system is powered by Grammatical Framework.

---

[1] `http://coherentknowledge.com`
[2] `http://gem.med.yale.edu/ergo/default.htm`
[3] `https://ttsreader.com`

Technical Communications of the 34th International Conference on Logic Programming (ICLP 2018).
Editors: Alessandro Dal Palu', Paul Tarau, Neda Saeedloei, and Paul Fodor; Article No. 22; pp. 22:1–22:7
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Methodology

In this section, we describe the `nlgPhylogeny` system. Figure 1 shows the overall architecture of `nlgPhylogeny`. The main component of the system is the *GF generator* whose inputs are the ontology and the elements necessary for the NLG task (i.e., the set of linearizations, the set of pre-define conjunctives, the set of vocabularies, and the set of sentence models) and whose output is a GF program, i.e., a pair of GF abstract and concrete syntax. This GF program is used for generating the descriptions of workflows via the GF runtime API. The adapter provides the GF generator with the information from the ontology, such as the classes, instances, and relations.

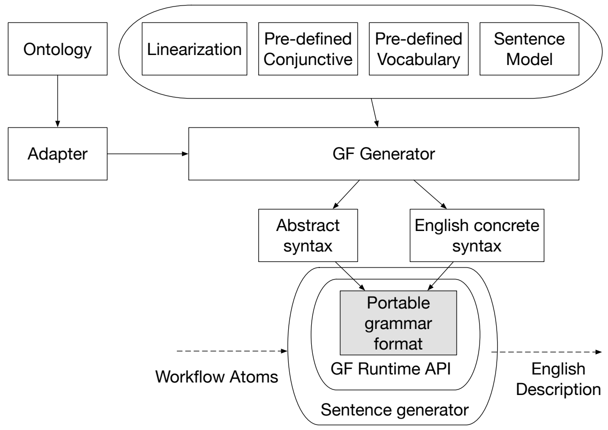

Figure 1 Overview of `nlgPhylogeny`

2.1 Web Service Ontology (WSO)

Phylotastic uses web service composition to generate workflows for the extraction/construction of phylogenetic trees. It makes use of two ontologies: WSO and PO. WSO encodes information about registered web services and their abstract classes. In the following discussion, we refer to a simplified version of the ASP encoding of the ontologies used in [3], to facilitate readability. In WSO, a service has a name and is associated with a list of inputs and a list of outputs. For example, the service which is named in ontology *phylotastic_FindScientificNamesFromWeb_GET* is an instance of the class *names_extraction_web*. The data that *phylotastic_FindScientificNamesFromWeb_GET* uses and produces are encoded by the following 3 atoms:

```
instance_operation_has_input_has_data_format(
  phylotastic_FindScientificNamesFromWeb_GET,
  resource_WebURL,
  url_format).
instance_operation_has_output_has_data_format(
  phylotastic_FindScientificNamesFromWeb_GET,
  resource_SetOfSciName,
  scientific_names_format).
instance_operation_has_output_has_data_format(
  phylotastic_FindScientificNamesFromWeb_GET,
  resource_SetOfNames,
  list_of_strings).
```

In regard the above atoms, the first argument is the name of the service, the second argument is the service input or output, and the last argument is the data type of the second argument.

The web service ontology of the Phylotastic project is exported to an ASP program (from its original OWL encoding) and an inference engine is provided for reasoning about classes, inheritance, etc. `nlgPhylogeny` employs this engine in identifying information related to the set of atoms whose description is requested by a user (e.g., What are the inputs of a service? What is the data type of an input x of a service y?).

2.2 GF generator

Each Phylotastic workflow is an acyclic directed graph, where the nodes are web services, each consumes some resources (inputs) and produces some resources (outputs). An example of the specification of workflow is as follows.[4]

```
occur_concrete(phylotastic_ExtractSpeciesNames_From_Gene_Tree_GET,1)
occur_concrete(phylotastic_ResolvedScientificNames_OT_TNRS_GET,3)
occur_concrete(phylotastic_GenerateGeneTree_From_Genes,0)
occur_concrete(phylotastic_GeneTree_Scaling,2)
```

This set of atoms is a partial description of the result of a web service composition process, as described in [3]. Intuitively, this set of atoms represents a plan consisting of 4 steps. At each step, a concrete instance of the service class named by the first argument of the atom `occur_concrete/2` is executed.

To generate the description of a workflow, we employ the framework described in [4]. This framework consists of three major processing phases: (1) document planning (content determination), (2) microplanning, and (3) surface realization. The document planning phase is used to determine the structure of the text to be generated. Based on the structure determined in the document planning phase, the microplanner makes lexical/syntactic choices to generate the content of the sentences, and the realization phase generates the actual sentences. In our work, we combine the microplanning and surface realization phase into a single phase due to the nature of the grammar definition and the capability of GF in sentence generation.

In the document planning step, we create—for each occurrence atom—a sentence which specifies the input(s) and output(s) of the service mentioned in the first argument of the atom. Optionally, to describe the service in more details, one or two more sentences about datatype of the service's inputs or outputs can be included. As we have mentioned in the previous subsection, the information about the inputs, outputs, and data types of the inputs and outputs of a service can be obtained via the ASP reasoning engine of the Phylotastic system. In general, we identify the following document planning structure:

relation: IDENTITY
argument_1: instance or class in ontology
argument_2: list of service inputs
argument_3: list of service outputs

(optional)
relation: IDENTITY
argument_1: name of input or output of service
argument_2: data type of argument_1

---

[4] For simplicity, we use examples which are linear sequences of services.

(optional)	
relation:	IDENTITY
argument:	actual data involved in the workflow

The document planning phase determines three messages for the sentence generation phase.

In the microplanning step, we focus on developing a GF generator that can produce a portable grammar format (**pgf**) file [1]. This file is able to encode and generate 3 types of sentences as mentioned above. The GF generator (see Fig. 1) accepts two flows of input data: The first one is the flow of data from the ontology which is maintained by an adapter. The *adapter* is the glue code that connects the ontology to the GF generator. Its main function is to extract classes and properties from the ontology. The second one is the flow of data from predefined resources that cannot be automatically obtained from the ontology—instead they require manual effort from both ontology experts and linguistic developers:

- A list of *linearizations*; these are essentially the translations of names of ontology entities into linguistic terms. This translation is performed by experts who have knowledge of the ontology domain. An important reason for the existence of this component is that some classes or terms used in the ontology might not be directly understandable by the end user. This may be the result of very specialized strings used in the encoding of the ontology by the ontology engineer (e.g., abbreviations), or the use of URIs for the representation of certain concepts. For example, the class *phylotastic_OTResolvedNames* can be meaningfully linearized to *OpenTree Name Resolution service.*
- Some *model sentences* which are principally Grammatical Framework syntax trees with meta-information. The meta-information denotes which part of syntax tree can be replaced by some *vocabulary* or *linearization.* As indicated above, we decided that each occurrence atom of a workflow will be described by at most three sentences. For example, in regards to the first message in the document planning structure, the generated sentence will have the inputs and the outputs of a service; the second message indicates a sentence about the data type of its first argument (input or output); the third message is about the actual data used during the execution of the workflow. However, the messages do not specify how many inputs and outputs should be included in the generated sentence. The structure of the sentence representing a service that requires one input and one output is different from the structure of sentence representing that a service that does not require any inputs. These variations in sentences are recorded in the *model sentence* component. An example of a model sentence, for the case of a service that has a single input is as follows:

  ```
  {
    "s": "mkS (mkCl subject_in p_in_1);",
    "placeholder": {
      "subject_in": ["input of subject", "subject’s input"]
    }
  }
  ```

- A list of *pre-defined vocabularies* which are domain-specific for the ontology. A *pre-defined vocabulary* is different from linearizations, in the sense that some lexicon may not be present in the ontology but might be needed in the sentence construction; the predefined vocabulary is also useful to bring variety in word choices when parts of a *model sentence* are replaced by the GF generator.
- A configuration of *pre-defined conjunctives* which depend on the document planning result. Basically, this configuration defines which sentences accept a conjunctive adverb

in order to provide generated text transition and smoothness.

To encode sentences, the GF generator defines 3 categories: Input, Output and Format in the abstract syntax.

```
abstract Phylo = {
  flags startcat = Message;
  cat
    Message; Input; Output; Format;
    ...
}
```

and the corresponding English concrete syntax:

```
concrete PhyloEng of Phylo = open
SyntaxEng, ParadigmsEng, ConstructorsEng in {
  lincat
    Message = S; Input = NP; Output = NP; Format = NP;
    ...
}
```

SyntaxEng, ParadigmsEng, ConstructorsEng are GF Resources Grammar libraries which provide some constructors for sentence components like Verb, Noun Phrase, etc.. in English.

The GF generator obtains information about the services (e.g., how many inputs/outputs has the service? what are the data types of the inputs/outputs? etc.) by querying the ontology (via the adapter). Each service will be mapped to several functions in GF:

- A function which encodes the meaning of the sentence used for describing the service. The GF generator will prefix the name of the service with *f_* to create this kind of function name.
- A function which encodes the meaning of each input. The GF generator will prefix the name of the input with *i_*.
- A function which encodes the meaning of each output. The GF generator will prefix the name of the output with *o_*.

Based on the number of inputs and outputs of a service, the GF generator determines how many parameters will be included in the GF abstraction function corresponding to the service. Furthermore, for each input or output of a service, the GF generator includes an *Input* or *Output* in the GF abstract function. As an example, the result of the encoding of the atom

```
occur_concrete(phylotastic_FindScientificNamesFromWeb_GET,1)
```

in the GF abstract syntax is

```
f_phylotastic_FindScientificNamesFromWeb_GET: Input -> Output -> Message;
i_resource_WebURL                           : Input;
o_resource_SetOfNames                       : Output;

```

Next, the GF generator looks up in the *sentence models* a model syntax tree whose structure is suitable for the number of inputs and outputs of the service. If such syntax tree exists, the GF generator will replace parts of the syntax tree with the GF service input and output functions, to create a new GF syntax tree which can be appended in the GF concrete function. The functions in the abstract syntax corresponds to the following functions in the GF concrete syntax:

```
f_phylotastic_FindScientificNamesFromWeb_GET i_resource_WebURL
o_resource_SetOfNames =
mkS and_Conj
    (mkS (mkCl phylotastic_FindScienticNamesFromWeb_GET_in
               (mkV2 "require")
               i_resource_WebURL))
    (mkS (mkCl phylotastic_FindScienticNamesFromWeb_GET_out
               (mkV2 "return" )
               o_resource_SetOfSciName ));

i_resource_WebURL = mkNP(mkCN (mkN "webURL"));
i_resource_SetOfNames = mkNP(mkCN (mkN "asetof names"));
```

The above functions consist of several syntactic construction functions which are implemented in the GF Resources Grammar library:

- mkN which creates a noun from a string;
- mkCN which creates a common noun from a noun;
- mkNP which creates a noun phrase from a common noun;
- mkV2 which creates a verb from a string;
- mkCl which creates a clause. Clause can be constructed from sequence of a noun phrase, a verb and another noun phrase (NP V2 NP);
- mkS which creates a sentence. Sentence can be constructed from a clause (Cl) or from 2 other sentences and a conjunction word (and_Conj S S).

From the abstract and concrete syntax built by GF generator, the atom `occur_concrete(phylotastic_FindScientificNamesFromWeb_GET,1)` is translated into the sentence

The input of phylotastic_FindScientificNamesFromWeb_GET is a web link and its outputs are a set of species names and a set of scientific names.

We use the same technique to encode the other types of sentences indicated by the document planning structure.

3 Discussion and future works

To the best of our knowledge, we found the work in [2] that reports on generating natural language text from class diagrams highly related to what we are doing. In [2], authors developed a system to generate specifications for UML class design. The difference between our work and [2] is the design of the system to employ automation on text generation for a given ontology under some assumptions.

From our case study we have identified two directions of future work that we find interesting. The first direction is to generate descriptions from annotations in ontology. We observe that the annotations play an vital role in ontology development in the sense of recording notes and explanations about concepts. Ontology developers usually use annotations to define the concepts and to describe relations between the concepts in the ontology, so that they employ reusablitity of the ontology. It is possible to apply natural language processing techniques to extract information from the annotation and tie that information with which concept or relation the annotation describes to re-generate text when needed. We believe that extracting and re-generating process is useful for query-answer system and information retrieval system since the process reduces the effort of system developers to create a module to explain the result of query.

The second direction is to make more use of the Grammatical Framework. We also want to make more of GF's capacity for several concrete languages to share the same abstract syntax. In other words, given an annotated ontology, we would like to generate explanations in multiple languages for a query.

References

1 Krasimir Angelov, Björn Bringert, and Aarne Ranta. Pgf: A portable run-time format for type-theoretical grammars. *Journal of Logic, Language and Information*, 19:201–228, 2010.

2 Hakan Burden and Rogardt Heldal. Natural language generation from class diagrams. In *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation (MoDeVVa 2011), Wellington, New Zealand, ACM*, 2011.

3 Thanh H. Nguyen, Tran Cao Son, and Enrico Pontelli. Automatic web services composition for phylotastic. In *Practical Aspects of Declarative Languages - 20th International Symposium*, pages 186–202, 2018. URL: `https://doi.org/10.1007/978-3-319-73305-0_13`, `doi:10.1007/978-3-319-73305-0_13`.

4 Ehud Reiter and Robert Dale. *Building natural language generation systems*. Cambridge university press, 2000.