International Journal of Semantic Computing
Vol. 8, No. 3 (2014) 335–384

DOI: 10.1142/S1793351X14500068

World Scientific
www.worldscientific.com

Natural Language Generation from Graphs

Ngan T. Dong
School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99163, USA
takura247@gmail.com

Lawrence B. Holder
School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99163, USA
holder@wsu.edu

The Resource Description Framework (RDF) is the primary language to describe information on the Semantic Web. The deployment of semantic web search from Google and Microsoft, the Linked Open Data Community project along with the announcement of schema.org by Yahoo, Bing and Google have significantly fostered the generation of data available in RDF format. Yet the RDF is a computer representation of data and thus is hard for the non-expert user to understand. We propose a Natural Language Generation (NLG) engine to generate English text from a small RDF graph. The Natural Language Generation from Graphs (NLGG) system uses an ontology skeleton, which contains hierarchies of concepts, relationships and attributes, along with handcrafted template information as the knowledge base. We performed two experiments to evaluate NLGG. First, NLGG is tested with RDF graphs extracted from four ontologies in different domains. A Simple Verbalizer is used to compare the results. NLGG consistently outperforms the Simple Verbalizer in all the test cases. In the second experiment, we compare the effort spent to make NLGG and NaturalOWL work with the M-PIRO ontology. Results show that NLGG generates acceptable text with much smaller effort.

Keywords: RDF; graph; natural language generation.

1. Introduction

Natural Language Generation (NLG) is a natural language processing task of generating natural language from a computer representation of data. The left side of Fig. 1 shows an example of data in computer representation (RDF graph) in which "v" refers to a vertex and "d" refers to a directed edge. Nodes and edge labels are Universal Resource Identifiers (URIs) or strings. This expression of a graph is not human friendly. A natural language generation system's task in this case is to translate these graphs into English text as illustrated on the right in Fig. 1.

The Semantic Web is an increasing effort of converting the existing set of Web documents into a giant graph of data. This representation brings a number of

RDF Graph:	Nat Lang Text:
v 1 http://www.co-ode.org/roberts/family-tree.owl#A	A, B and C are people. B and C are A's children.
v 2 http://www.co-ode.org/roberts/family-tree.owl#Person	
v 3 http://www.co-ode.org/roberts/family-tree.owl#B	
v 4 http://www.co-ode.org/roberts/family-tree.owl#Person	
v 5 http://www.co-ode.org/roberts/family-tree.owl#C	
v 6 http://www.co-ode.org/roberts/family-tree.owl#Person	
d 1 2 http://www.w3.org/1999/02/22-rdf-syntax-ns#type	
d 3 4 http://www.w3.org/1999/02/22-rdf-syntax-ns#type	
d 5 6 http://www.w3.org/1999/02/22-rdf-syntax-ns#type	
d 1 3 http://www.co-ode.org/roberts/family-tree.owl#hasChild	
d 5 1 http://www.co-ode.org/roberts/family-tree.owl#hasParent	

Fig. 1. Example input/output of NLGG system. *Left*: Machine representation of RDF graph. *Right*: Natural language text describing RDF graph.

benefits, including but not limited to reusing, sharing and intelligent searching. The Resource Description Framework (RDF) is a language to describe information on the Semantic Web. Web resources are represented as RDF triples. A set of RDF triples form an RDF graph. Generally, an RDF graph is equivalent to a directed graph. Further details about the Semantic Web and RDF will be given in Sec. 3.

The announcement of schema.org by Yahoo, Google, and Bing; the ongoing Linked Open Data project led by the World Wide Web Consortium (W3C); along with a number of other research efforts from the government, industry and academic institutions, have resulted in a significant increase in the amount of data available in RDF format. Yet RDF triples are computer friendly and thus sometimes are hard for the non-expert user to understand. What is more, since the predicate in a RDF triple can be anything, graph visualization sometimes cannot solve the problem. Also in some systems that do semantic search or anomaly detection, the output should be in the form of natural language rather than graphic. Therefore, we believe a natural language generation system, which expresses a small RDF graph into English text, is needed.

Research in Natural Language Generation applied to the Semantic Web is in its initial stages. NaturalOWL [1], a domain independent sentence generator from RDF graph [2], and some work in verbalizing OWL ontologies [3–5] are some current works in the field. However, their problem statements are different from the above one. We propose a general natural language generation from graphs (NLGG) approach to bridge the gap between Semantic Web representation and natural language so that people who are not familiar with RDF can also access the information and knowledge for various purposes.

Figure 2 illustrates a potential use case of NLGG in which a large RDF graph is processed to produce several small RDF graphs that are converted to English text. The input graph can be, for example, a social network graph extracted from Facebook or a citation network or web logs of user activities from a website, in which

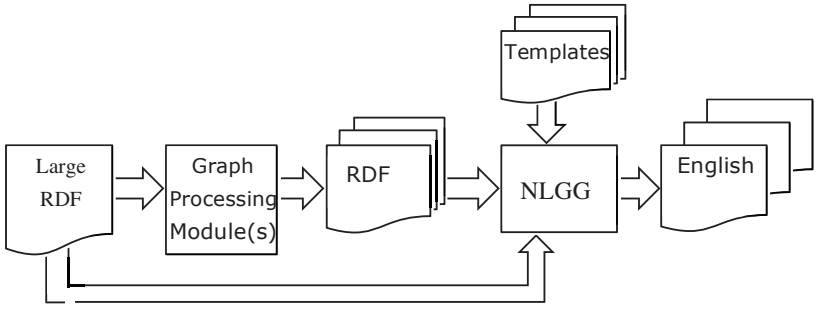

Fig. 2. An example of NLGG use case in which graph processing is used on a large RDF graph to produce multiple smaller RDF graphs (e.g. subgraph patterns, anomalies) that are then presented to the user as English text.

nodes can be people, agents, or groups; and edges are relationships between them. The graph processing might consist of subgraph pattern miners or anomaly detectors, which produce several small RDF subgraphs. The set of RDF graphs, along with the original large graph and template information, are passed into NLGG as the knowledge base. NLGG pre-processes the large graph to eliminate information related to individuals such as their names, ages, jobs, and relationships with others. For each small RDF graph, NLGG generates a corresponding piece of English text for output to the user. The main focus of our work is to verbalize a small set of RDF triples into English text. By saying verbalize, we mean to focus on the way in which a set of facts is presented to the user. We use a semantic reasoner to help generate better text. A skeleton ontology is used instead of the full original graph to minimize the system's running time. We minimize the manual input of lexical information into the system to make it more portable and cost effective. Compared with a template-base NLG system like NaturalOWL, NLGG is much easier to set up. The system's capability in generating cohesive text with connecting words is still limited. To the best of our knowledge, this is the first work in the field to deal with the problem of verbalizing a small subset of a RDF graph that involves information about multiple individuals/classes.

Our goal is to help non-experts understand the content of a small RDF graph. A graph visualizer can fulfill the task by generating a graph as depicted in Fig. 3. Graph visualization is preferred in some cases because it is intuitive. Also, visualization requires no handcrafted template information that most of the time will be required by natural language generation. However, we argue that a natural language generation engine for RDF graphs is still needed for three main reasons. First, RDF predicates can be any string, and that string sometimes is not expressive or familiar enough for the user to understand the meaning. A graph visualizer can only render the connection between nodes. It cannot help the user understand the meaning of a RDF triple or RDF predicate, and thus, fails to fulfill the task. Second, for those systems that do anomaly detection, the output of the system should not be a graph

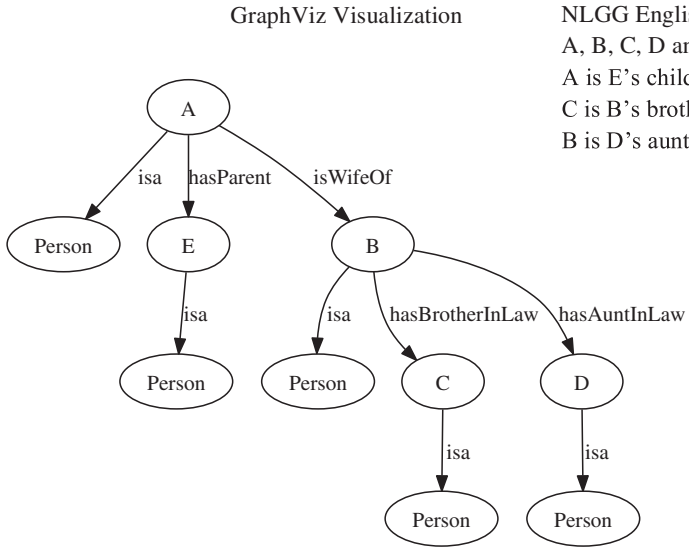

Fig. 3. Visualization of small graph pattern using GraphViz [6] (left). English text expression of graph using NLGG (right).

with nodes and edges. Instead, it should be a spoken or written warning message in natural language. Third, visually-impaired users, or users in an environment where visual focus must be elsewhere, would benefit from a spoken description of the graph based on a natural language expression. Therefore, we believe that though a graph visualizer can sometimes help solve the problem, there is still a need for a natural language generation capability for expressing graphs.

In the remainder of this paper, Sec. 2 describes natural language generation systems, their applications, architecture and type of input. Section 3 discusses the Semantic Web and related concepts. Section 4 explains the NLGG architecture step by step. Section 5 presents the experimental results. Section 6 discusses conclusions and future directions.

2. Natural Language Generation

Natural Language Generation (NLG) is a fascinating area of research with many real-world applications. Most current NLG systems are used either to present information to users, or to partially automate the production of routine documentation. There is no consensus among NLG research communities for a standard input format, system structure or evaluation techniques. The knowledge models vary from system to system. This makes most existing NLG systems heavily domain dependent and hard to reuse. This section will give a brief overview of Natural Language Generation systems, their input, system architectures, applications and evaluation techniques.

2.1. *Definition*

Natural Language Generation (NLG) is the natural language processing task of generating natural language from machine readable data such as numerical values, logical form, XML, etc., to meet specified communicative goals. In other words, an NLG system serves as a translator that converts a computer based representation into natural language representation. The communicative goals represent the purpose of the text to be generated. A weather reporter's communicative goal can be to summarize the weather data for a specific month. The communicative goals of ontology verbalizers, such as NaturalOWL, are to generate a description about an entity or a concept. And the goal of NLGG is to verbalize a set of given RDF triples.

As regards to the knowledge foundation, natural language generation is closely related to natural language understanding [7]; both of them are concerned with computational models of language and their use. In regard to the internal operations, natural language generation may be viewed as the opposite of natural language understanding. Natural language generation is the process of mapping internal computer representations of information into human language; whereas, natural language understanding is the process of mapping human language into internal computer representations.

NLG research involves knowledge of artificial intelligence, cognitive science and human-computer interaction. Over the past two decades, NLG has received considerable attention among research communities. Over 385 NLG systems, with many real-world applications, have been documented so far at the NLG Wiki (www.nlg-wiki.org). In the long term, NLG plays an important role in human-computer interaction and will help provide much richer interaction with machines than today.

2.2. *Input to NLG systems*

As summarized by Reiter and Dale [7], inputs to a NLG system consist of four components: a knowledge base, a communicative goal, a user model and a discourse history. The knowledge base contains information about the system's domain. Both the content and representation of the knowledge base are highly application-dependent. A weather forecaster's knowledge base can be a database of numerical sensor measurements. A museum guide NLG system like NaturalOWL [1] uses an annotated OWL ontology about museum artifacts as the knowledge base. And those of ontology verbalizers like the Sydney OWL syntax [3], the Rabbit controlled natural language [4], and the Attempto Controlled English (ACE) [5], are OWL ontologies which contain formal definitions about classes, individuals and properties.

NLGG's knowledge base is an OWL ontology which contains only information about classes and properties, but not individuals, plus simple template information for those classes and properties. There is a trade-off between the quality of the text generated and the amount of handcrafted domain-dependent information input into the system. Our objective is to make NLGG as general as possible. And thus, the type of template required is simple. However, that places some limitations on quality

of the output text. For example, NLGG cannot generate referring expressions or use pronouns and other references to make the text more fluent and natural. More details about NLGG's knowledge base will be given in Sec. 4.1.

In general, there is no formal characterization of what a knowledge base should look like. The wide variation among existing NLG systems' input knowledge bases is a result of the differences in a system's text planning, localization and realization strategies; and thus make existing NLG applications highly domain dependent. The richer semantics that the knowledge base contains, the more power the system has in generating natural, customized, fluent text.

A communicative goal defines the purpose of the generated text. Different NLG systems have different sets of communicative goals (e.g. automated generation of reports, letters, jokes, help messages, customized introductions, etc.). A weather forecaster's communicative goal is to summarize the weather data for a specific month. NaturalOWL's communicative goal is to summarize information related to an artifact as regards to the user model and discourse history. An ontology verbalizer's goal is to map OWL representation into natural language. NLGG's communicative goal is to verbalize a set of small RDF triples.

A user model refers to the characteristic of an NLG system's targeted user. Mono-user model systems like ontology verbalizers, or our current system (NLGG), which only support one type of user, are not concerned about a user model. They do not need to customize the generated text according to the current user's domain knowledge. But multi-user model systems like NaturalOWL do need to take the user model into account. A user model in this case defines the user's understanding of the domain. And this, in turn, affects the level of details included in the generated text.

A discourse history is simply a log of what information has already been communicated so far. In other words, it keeps track of what facts have already been outputted to the user. This helps the NLG system avoid repetition as well as flexibly use pronouns or relative clauses. For interactive systems, discourse history plays a central role in generating intelligent and natural text. But for ontology verbalizers that were mentioned above and the NLGG, where the content of the output text is intended to stay the same for all users, discourse history is rarely given much account.

2.3. *NLG applications*

The highest motivation behind Natural Language Generation technology is that computer systems' representations of information usually require a considerable amount of expertise to interpret. Therefore, there is often a need for systems that can present such data in an understandable form to non-expert users. NLG systems have been effectively used for the following tasks.

- Generate textual weather forecasts from numerical data, and summary reports, as in [8–11]. These works often involve the task of summarizing information from a set of data. Research in the field indicates that users sometimes prefer

computer-generated forecasts to human written ones [9], partly because computer forecasts used more consistent terminology.

- Explain medical information in a patient-friendly way, as in [12–14]. These systems aim at providing decision support to medical professionals. Also they help keep family members informed about the patient's condition.
- Describe concepts in the knowledge base. Natural OWL [1] is a natural language generation system that has been used to describe museum artifacts. The system supports multilingual language and multi-user model text generation.
- Develop interactive question answering [15], generate adaptive stories [16], and dynamic drama plots [17]. The unique characteristic of these systems is that they are capable of handling the user interaction in the generation process.
- Present geo-referenced data available for visually impaired population, as in [18]. Geo-referenced data is often in the form of maps and so is hard or impossible for users with visual impairment to access. Thomas and Sripada's system takes this kind of data representation as input and generates textual summaries which can be read out loud to users via screen reader software and thus help overcome the visual impairment issue.

In addition, NLG technology can be used to build authoring aid systems that help people create routine documents. Examples of using NLG as an authoring aid include as follows.

- Helping domain experts to encode their knowledge directly, by interacting with a feedback text generated by the system, as the work proposed by Power [19]. Power developed the What You See Is What You Meant (WYSIWYM) system that is a NLG framework that allows domain expert users to create and manipulate knowledge representations, such as those required by the Semantic Web using a natural language interface, thereby editing the formal language without learning it. Some works on OWL ontology verbalizers like the Sydney OWL syntax [3], the Rabbit controlled natural language [4], the Attempto Controlled English (ACE) [5] are attempts to make OWL ontologies accessible to people with no training in formal methods. Further details about these ontology verbalizers will be given at the end of this section. Information about OWL ontologies and the Semantic Web will be presented in the next section.
- Helping personnel officers produce multilingual job advertisements, as in [20]. Job adverts are stored as language-independent schemas in the database. Users can search the database in their own language and get customized summaries of the job ads. This NLG engine used canned text, templates, and grammar rules to produce texts and hypertexts.
- Helping technical authors produce instructions for using software, as in [21, 22]. The technical author specified the instructions of different tasks in a non-linguistic representation. NLG engines will express these content specifications as instructional texts that conform to the style of software user manuals.

2.4. *NLG architecture*

Though different NLG systems have different architectures and cover different tasks, Ehud Reiter in [23] summarized that almost all applied NLG systems perform the following three tasks.

- *Content Determination and Text Planning*: Decide what information should be communicated to the user (content determination) and how this information should be rhetorically structured (text planning). These tasks are usually done simultaneously.
- *Sentence Planning*: Decide how the information will be split among individual sentences and paragraphs, and what cohesion devices (e.g. pronouns, discourse markers) should be added to make the text flow smoothly.
- *Realization*: Generate the individual sentences in a grammatically correct manner.

This set of tasks might not cover all the possible problems associated with a natural language generation system. Also different NLG engines might address them differently and/or in a different order. However, all these issues must be dealt with one way or another by a complete NLG system. In the rest of this section, different ways of performing each of these tasks shall be briefly discussed.

2.4.1. *Content determination and text planning*

Content determination decides what information to communicate in the text. In most NLG systems, NaturalOWL for example, the set of information related to an individual in the ontology contains information about its properties, the individuals or classes that it connects to, and information related to those individuals or classes. Generally, not all this information should be output in the text since there might be repetition and also the user might have already known something. So the task of the content determination module in that case is to decide what should be included in the output text and what should not. In other words, it is responsible for selecting some appropriate subset of the available information.

However, the content determination module of NLGG is slightly different. Because the set of facts communicated in the text is fixed in the NLGG system; its content determination module only decides how a fact should be output to the user by flexible use of inverse, symmetric properties as given in the knowledge base. For example, instead of output "*C hasParent A*", NLGG chooses "*A hasChild C*" so that it can be aggregated with another fact "*A hasChild B*" in the input to produce the sentence "*B and C are A's children*". Further details about this shall be given in Sec. 4 when the NLGG system architecture is presented.

Text planning organizes the information into a rhetorically coherent structure. These tasks can be done at many different levels of sophistication, ranging from hard-coded, rule-based to special schema or special language. The more sophisticated the system's content determiner and text planner, the more flexibility the system

engineer has in generating rhetorical and sentence structure. NaturalOWL organizes facts according to distance from the individual or class being described. Facts that have the same distance are organized according to handcrafted ordering annotations in the knowledge base. NLGG does some slightly different text planning jobs. Since NLGG has no notion of focus point (i.e. the individual or class being described as in NaturalOWL), NLGG groups facts into three types: class definitions, individuals' attributes and individuals' relationships; and organizes them in that order. Facts in the same category are organized by priorities (which are included in the knowledge base). Lack of focus point might make the text less cohesive or look unnatural. However, for a small piece of text, we find no such problems in this document structuring strategy.

2.4.2. *Sentence planning*

Sentence planning includes the following three operations.

- Conjunction and other aggregation. For example, transforming (a) into (b):

  (a) Sam has high blood pressure. Sam has low blood sugar.
  (b) Sam has high blood pressure and low blood sugar.

- Pronominalization and other reference. For example, transforming (c) into (d):

  (c) I just saw Mrs. Black. Mrs. Black has a high temperature.
  (d) I just saw Mrs. Black. She has a high temperature.

- Introducing discourse markers. For example transforming (e) into (f):

  (e) If Sam goes to the hospital, he should go to the store.
  (f) If Sam goes to the hospital, he should also go to the store.

The common theme behind these operations is that they do not change the information content of the text, but they do make it more fluent and readable. Look at the text generated by NaturalOWL in Fig. 4. We can see that it does a good aggregation job as well as flexibly uses pronouns and reference (e.g. "this particular aryballos",

> **exhibit24:** This is an aryballos, a kind of vessel. An aryballos was a small spherical vessel with a narrow neck, in which the athletes kept the oil they spread their bodies with. This particular aryballos was found in the Heraion of Delos and it is currently exhibited in the Archaeological Museum of Delos. It was created during the archaic period. The archaic period was the time during which the Greek ancient city-states developed and it covers the time between 700 B.C. and 480 B.C. This aryballos was decorated with the black-figure technique. In the black-figure technique, the silhouettes are rendered in black on the pale surface of the clay and details are engraved.

Fig. 4. Example output produces by the NaturalOwl system.

"it", and "this aryballos" were used interchangeably to refer to the item being described (i.e. exhibit24)). NLGG's sentence planning module only covers the first task out of the three listed at the beginning of this section. That means NLGG's text planning (or document structuring) will only aggregate sentences and does not support the use of pronominalization or other reference and neither does it consider discourse markers. This is partly due to the simple type of template used as well as the lack of focus point in the generated text. Further details about NLGG's aggregation strategies shall be given in the next section.

To sum up, sentence planning is fundamental if the text's fluency is important, and it should look like it was written by a human (which is usually the case for a museum guide agent, for example). If it does not matter that the text sounds stilted and was obviously produced by a computer, then it may be possible to de-emphasize sentence planning, and perform minimal aggregation, use no pronouns, etc. Otherwise, a good job of sentence planning is essential.

2.4.3. *Realization*

A realizer generates individual sentences, typically from a deep syntactic representation. The realizer needs to make sure that the rules of English are obeyed, including the following.

- Point absorption and other punctuation rules. For example, the sentence "I saw Helen Jones, my sister-in-law" should end in ".", not ",.".
- Morphology. For example, the plural of "box" is "boxes", not "boxs".
- Agreement. For example, "I am here" instead of "I is here".
- Reflexives. For example, "John saw himself" instead of "John saw John".

There are numerous linguistic formalisms and theories that can be incorporated into an NLG realizer. Among those, the SimpleNLG [24] is a general NLG system that serves as a "realization engine" and has been used successfully in a number of projects. NLGG uses SimpleNLG to do the realization jobs. Looking at the output text in Fig. 1, we can see that punctuation rules and morphology (e.g. use "children" instead of "childs") have been successfully followed in the output text.

2.5. *NLG systems evaluation*

The two most popular NLG evaluation methods are those based on task performance and human judgments/ratings. Task-based evaluation evaluates an NLG system by assessing human performance on a task in the application domain. This type of evaluation is helpful when NLG developers need to persuade someone with different expertise (for example, a doctor or a psychologist) about the usefulness of the NLG system. However, it can be expensive and time-consuming to carry out. Human ratings and judgments ask human experts to judge/rate the generated text on some scales. Human-based evaluation tends to be quicker and cheaper to carry out, and

thus seems to be preferred over task-based evaluation. Another reason that makes human judgments more prominent is that it is not always possible to conduct meaningful task-based evaluations on some NLG systems. Further details about these methods will be given below. Some other evaluation efforts that focus on NLG individual components will also be briefly presented at the end of this subsection.

2.5.1. *Task-based evaluation*

Task-based evaluation looks at human users' success or failure in performing a task in the application domain using the NLG system in question. STOP [25] is a NLG system for generating personalized smoking-cessation letters. It was evaluated on the basis of medical effectiveness. The authors sent a group of 2000 smokers either STOP-generated letters or one of two kinds of control letters, and measured how many smokers in each group managed to quit smoking. SKILLSUM [11], which generates feedback reports from literacy assessments, was evaluated on the basis of educational effectiveness. They gave 200 assessment takers either SKILLSUM texts or control texts, and measured whether they increased the accuracy of self-assessments of their literacy skills. The Generating Instructions in Virtual Environments (GIVE) [26] represents another set of current attempts in task evaluations. GIVE is a community-shared task. Users connect to the NLG systems via the Internet. The NLG systems' tasks are to generate real-time instructions that help users accomplish a "treasure hunt" task in a virtual 3D environment. User performance is logged and used to analyze the quality of the generated instructions. Three installments of GIVE have been run so far from 2008 to 2012. GIVE is one of the largest ever NLG evaluation efforts in terms of the number of experimental subjects. The GIVE organizing committee claims that their results are agreeable but more detailed than those that were obtained from lab-based evaluations.

Task-based evaluations have traditionally been regarded as the most meaningful kind of evaluation in NLG, especially in the context where the evaluation needs to convince people in other communities about the effectiveness/usefulness of the NLG applications (e.g. psychologists and physicians). However, they can be expensive and time-consuming. Reiter and Belz [27] summarized that the STOP evaluation cost UK£75,000, and required 20 months to design, carry out, and analyze; the SKILLSUM and one of the BabyTalk systems cost UK£20,000 over six months. These studies cost a great deal of money and time, which cannot be afforded by every NLG system.

2.5.2. *Evaluation based on human judgments or ratings*

Evaluation based on human judgments or ratings asks human experts to judge the quality of generated texts on an n-point rating scale. This type of evaluation is the most popular way of evaluating NLG systems, because it is relatively cheap and

quick to carry out. Also it can provide insights into the general strengths and weaknesses of the NLG technology.

This methodology was first used in NLG by Lester and Porter [28]. They asked eight domain experts each to rate 15 texts on a number of different dimensions: overall quality and coherence, content, organization, writing style, and correctness. Some of the texts were human written and some were computer-generated, but the judges did not know the origin of specific texts they read. Many more such evaluations have been performed since, often with fewer dimensions. A variation of this technique is to show subjects different versions of a text, and ask them which one they prefer. We use this type of evaluation to evaluate the text generated by the NLGG system. Output texts from a Simple Verbalizer are compared with the results generated by NLGG. The Simple Verbalizer uses no template nor requires any manual syntactic or lexical information. It simply reads in RDF triples and for each triple uses simple rules to generate English text. Further details about the Simple Verbalizer based NLGG evaluation are given in the next section.

2.5.3. *Other works*

Besides the above two methodologies, NLG researchers also pay attention to the evaluation of the performance of individual components inside an NLG system. Yeh and Mellish [29], for example, evaluated the performance of a referring expression generation algorithm by comparing its decision about which type of referring expression was appropriate in a given context to the decision made by human writers who were given the same problem.

The surface realizer shared task [30] is an ongoing research effort in which they manually evaluate and compare the performance of different surface realizers, whose inputs are the same, in terms of Clarity, Readability and Meaning Similarity. The RDF-content selection shared task challenge [31] is another recent community effort on content selection from a common semantic-web format input. Participants were asked to select the content communicated from a large volume of RDF triples about reference biographies. The selected triples are evaluated against a gold triple selection set using standard quality assessment metrics. Since those works can only evaluate individual parts inside an NLG system, there is some concern that the results of such evaluations may not be meaningful.

Overall, human ratings are currently the most popular evaluation technique in NLG. There are, as yet, no definitive answers as to how NLG systems should be evaluated.

2.6. *Difference between NLGG and existing works*

To summarize our discussion about NLG systems, we present a comparison between our NLGG system and some existing systems, which include NaturalOWL, Ontology Verbalizers, and the domain independent sentence generator.

2.6.1. *NLGG versus NaturalOWL*

NaturalOWL [1] is a multilingual natural language generation system that produces personalized descriptions of individuals and classes starting from a linguistically annotated ontology (the M-PIRO's museum ontology) (Fig. 5). NaturalOWL is a template-based system that requires manual input for various types of templates. Besides lexicon information for classes, instances and properties, it also requires information about user models, micro-plans with regards to user types, canned texts (a fixed string associated with a class or properties) and aggregation rules. Rich lexical and template information makes the knowledge base creation process of NaturalOWL expensive and time consuming. However, it enables the system to generate more complex types of text, which contains referring expressions and is customized according to user models. In addition, referring expressions generation helps avoid repetition and thus makes the output text more coherent and flow smoothly. Figure 5 presents information about one individual of class aryballos (exhibit24) stored in the system's ontology. Figure 4 shows NaturalOWL generated text for that individual. Figure 6 gives an example of a lexicon entry to a class which is much more complicated than that of NLGG (which is given in Table 1, Sec. 4.1).

So NLGG and NaturalOWL are both template-based NLG systems whose inputs contain an annotated OWL ontology. However, there are three main differences between NLGG and NaturalOWL. First, their types of output text are different. NaturalOWL can produce multilingual personalized text with referring expressions while NLGG can only generate a fixed small simple piece of English text for a given set of facts. Second, the type of template information required by NLGG is much simpler than that of NaturalOWL. We can see a significant difference by comparing Fig. 6 and Table 1 (Sec. 4.1). In short, NLGG requires only one type of template for classes and properties and that type of template is much simpler than that required by NaturalOWL. And finally, the content determination and document structuring of the two systems are fundamentally different. While NaturalOWL content determination module's task is to collect appropriate facts associated with a given class or individual, NLGG's content determination module tries to switch the way a set of given facts will be rendered to the user. NaturalOWL's document structuring

```
<aryballos  rdf:ID = "exhibit24">
    <creation-time  rdf:resource = "#sevencenturylateBC"/>
        <creation-location  rdf: resource = "#archaelogical-delos"/>
        <exhibit-style  rdf:resource = "#corinthian-style"/>
        <location-found  rdf:resource = "#Iraion-Delos"/>
        <painting-technique-used  rdf:resource = "#black-figure-technique"/>
        <exhibit-story  rdf:resource = "#exhibit24str"/>
        <creation-period  rdf:resource = "#archaic-period"/>
</aryballos>
```

Fig. 5. Entry for the exhibit24 artifact in the M-PIRO's museum ontology.

```
<owlnl:owlClass rdf:about = http://.../mipro.owl#vessel>
    <owlnl:hasNP rdf:resource = "#vessel-NP"/>
</owlnl:owlClass>
<owlnl:NP rdf:ID = "vessel-NP">
    <owlnl:LanguagesNP rdf:parseType = "Collection">
        <owlnl:GreekNP>
            <owlnl:countable>yes</owlnl:countable>
            <owlnl:num>singular</owlnl:num>
            <owlnl:gender>neuter</owlnl:gender>
            <owlnl:singular>
                <owlnl:singularForms>
                    <owlnl:nominativexml:lang = "el">…</owlnl:nominativexml:lang = "el">
                    <owlnl:genitivexml:lang = "el">…</owlnl: genitivexml:lang = "el">
                    <owlnl:accusativexml:lang = "el">…</owlnl: accusativexml:lang = "el">
                </owlnl:singularForms>
            </owlnl:singular>
            <owlnl:plural>
                <owlnl:pluralForms>
                    <owlnl:nominativexml:lan g = "el">…</owlnl:nominativexml:lang = "el">
                    <owlnl:genitivexml:lang = "el">…</owlnl: genitivexml:lang = "el">
                    <owlnl:accusativexml:lang = "el">…</owlnl: accusativexml:lang = "el">
                </owlnl: pluralForms>
            </owlnl: plural>
        </owlnl:GreekNP>
        <owlnl:EnglishNP>
            <owlnl:countable>yes</owlnl:countable>
            <owlnl:num>singular</owlnl:num>
            <owlnl:gender>neuter</owlnl:gender>
            <owlnl:singular>vessel</owlnl:singular>
            <owlnl:plural>vessels</owlnl: plural >
        </owlnl: EnglishNP >
    </owlnl:LanguagesNP>
</owlnl:NP>
```

Fig. 6. A lexicon entry for a class in the NaturalOwl system.

strategies are based on the distance from the individual/class being described, while that of NLGG is based on the type of facts. In summary, though the two systems are both template-based NLG systems using OWL ontologies, they have different types of input (communicative goals, knowledge base (i.e. template)), and thus support different types of output text.

2.6.2. *NLGG versus domain independent sentence generation from RDF representations*

Sun and Mellish [2] proposed a generic method for independent sentence generation from RDF representations based on the observation that RDF triples contain rich implicit linguistic information. Input to the system is a list of connected RDF triples, and they assume that the number of RDF triples is not too much for a single sentence (no more than 10 triples). For each triple, they use WordNet to parse it into tokens. After that, the syntax and linguistic information of those tokens is guessed based on a set of handwritten rules. Their system lexicalizes a property as a Verb Phrase (VP) or a Prepositional Phrase (PP). Examples of their type of phrases in the paper are restricted to the form of "*has something*" for a VP and similarly, "*with something*" for a PP. Their generation strategies go through a number of expensive operations involving the creation of a tree, lexicalizing, shifting, balancing and aggregating to have a complete lexicalized syntax tree. They placed a number of restrictions in their algorithm, and a formal evaluation has not been given yet. Figure 7 gives an example of the input and output from the domain independent sentence generator. Figure 8 gives an example of the text that might be generated by NLGG given the input graph as in Fig. 7. We can observe several differences between the two systems' output texts. NLGG cannot use pronouns while the domain independent sentence generator uses the pronoun "it" in the second sentence to replace "LongridgeMerlot" in the first sentence. NLGG can verbalize "hasMaker" as "is made by" while the sentence verbalizer cannot.

So there are two fundamental distinctions between Sun and Mellish's work and ours. First, they support different kinds of RDF graphs. We do not restrict the size of the input RDF graph to 10 triples nor the content to only one individual or one class as that of the domain independent sentence generator. Second, the domain independent sentence generator does not require manual input of lexical information while NLGG does require it. Further information about NLGG's template will be given in Sec. 4.1.

RDF Triples:
(LongridgeMerlot, RDF:type, Merlot)
(LongridgeMerlot, locatedIn, NewZealandRegion)
(LongridgeMerlot, hasMaker, Longridge)
(LongridgeMerlot, hasSugar, Dry)
(LongridgeMerlot, hasFlavor, Moderate)
(LongridgeMerlot, hasBody, Light)

English Text:
LongridgeMerlot is a kind of Merlot with dry sugar, moderate flavor and light body. It is located in New Zealand Region and it has maker Longridge.

Fig. 7. Example input and output from the domain independent sentence generator.

NLGG English Text:
Longridge Merlot is a Merlot. Longridge Merlot locates in New Zealand Region. Longridge Merlot is made by Longridge. Longridge Merlot has dry sugar, moderate flavor and light body.

Fig. 8. Output that might be generated from NLGG for the input graph in Fig. 7.

Domain Independent NLG is the most economic and generic way of generating natural language text. However, we argue that this approach is problematic for four main reasons. First, using only simple rules to find neighborhoods in the tree and placing a number of restrictions on the tree size and structures limits the system's capability. Second, their system performance depends heavily on WordNet's analysis result as well as the coverage of their handwritten rules which were extracted based solely on the set of RDF triples in their set of ontologies. WordNet has errors and there is no threshold for that error rate, especially when the predicate in an RDF triple can be any string and can be used in any structure. Also, parsing and guessing syntactic information for a set of tokens reintroduces another set of problems facing today's Web: interpreting natural language automatically. Thus, there is no guarantee about the system's performance on a new ontology that is different from what they have observed so far. Third, lexicalizing a property into "has something" or "with something" is too restrictive. It fails to render properties like "*bornIn*" or "*teach*", etc. And finally, their syntactic and lexical meaning guesser fails even in the cases where the predicate does follow their assumed forms (e.g. predicates start with verb or noun: the predicate "http://xmlns.com/foaf/0.1/maker" should be interpreted as "is made by"). From those observed restrictions, we believe that an NLG system for small RDF graphs is still needed, and that requiring simple template information to be manually input into the system is appropriate.

2.6.3. *NLGG versus ontology verbalizers*

It is a common belief that the mathematical nature of description logics makes it difficult for non-logicians such as domain experts to understand and author OWL-based ontologies [32]. Based on this, [3–5] proposed three controlled natural languages that aim at assisting non-OWL experts in building an ontology in OWL. A controlled language is a subset of natural language with constraints on grammar, lexicon and style. Ontology verbalizers like those mentioned earlier try to transform a hierarchy of classes, object properties, and data type properties into English text. They use a set of construction rules to define the syntax and another set of interpretation rules for dealing with ambiguity. Figure 9 gives an example of input and output from those ontology verbalizers where ACE, RAB, and SOS respectively refer to systems described in [3–5].

So the first difference between NLGG and these ontology verbalizers lies in the type of input into each system. While current work on ontology verbalizers only

OWL	SubClassOf(OWLClass(factory), ObjectSomeValuesFrom(ObjectProperty(has-part), ObjectIntersectionOf([ObjectSomeValuesFrom(ObjectProperty(has-purpose), OWLClass(manufacturing)), OWLClass(building)])))
ACE	For every factory its part is a building whose purpose is manufacturing
RAB	Every Factory has a part Building that has Purpose Manufacturing
SOS	Every factory has a building as a part that has a manufacturing as a purpose

Fig. 9. Example output from three ontology verbalizers: ACE, RAB and SOS.

supports concept definitions, NLGG allows relationships and attributes of individuals/classes to be included in the output text. What is more, the syntax rules and lexicalization rules are hard-coded into those ontology verbalizers, while in NLGG they are passed in as templates. That makes NLGG more flexible and easier to adapt to changes.

In summary, to the best of our knowledge, NLGG is the first work in the field that deals with the task of verbalizing a small, connected RDF graph. There are existing NLG works on RDF and OWL ontology, but we argue that they do not meet the need of our current problem statement. NaturalOWL only focuses on generating descriptions for an individual or a class, and the expensive annotation process makes it infeasible for a general NLG engine. Domain independent sentence generator is efficient and general, but it reintroduces several other hard problems that have not been solved yet. Also hard-coded rules and restrictions make the system only a novel approach that still needs more research effort before it can be used in real-world applications. As regards to the work on ontology verbalizers, their goals and scopes are different from NLGG. Thus, we believe that NLGG meets the need of a general, efficient NLG system for a specific group of Semantic Web applications in which output of the system is a small RDF graph.

3. The Semantic Web and OWL 2 Ontology

As we have previously mentioned, RDF triples are designed for computers and thus, hard for non-expert users to understand. Natural language generation in this case will help bridge the gap between the Semantic Web representation and natural language so that people who are not domain experts can still access information. The difficult to understand nature of the Semantic Web data is the main motivation behind our work. In this section, further details about that computational representation shall be discussed. We start with a brief overview about the Semantic Web and its applications. Next, we present an introduction to the OWL 2 Ontology language, RDF/XML syntax and the two types of RDF triples. And finally, an overview about Semantic reasoners will be given at the end of this section.

3.1. *The semantic web*

The Semantic Web is a collaborative movement led by the World Wide Web Consortium (W3C). The motivation behind the Semantic Web is to convert existing Web documents into a giant graph of semantically rich data that makes reuse, sharing, combining and searching information easier. Tim Berners-Lee originally expressed the vision of the Semantic Web as follows [33].

> "I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web — the content, links, and transactions between people and computers. A "Semantic Web", which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The "intelligent agents" people have touted for ages will finally materialize."

The Semantic Web has received more and more attention among research communities recently. Current research efforts include the announcement of Google Semantic Web search, the Linked Open Data community project, the annotation of Facebook, and Twitter OpenGraph. There is also an increasing research effort both from academic and commercial companies to integrate the Semantic Web into existing applications. In the past decade, the Semantic Web research community has been growing fast in size and scope. Large companies like Oracle, IBM, Adobe, Software AG and Yahoo have already offer business solutions based on this technology. The announcement of schema.org from Yahoo, Bing and Google has also facilitated the creation of more semantically rich data.

Semantic Web technologies have been used in a variety of domains, including but not limited to broadcasting [34], health care [35–37], search [38–40], public institutions [41], and eGovernment [42, 43]. The complexity and variety of applications referring to the Semantic Web is increasing every day.

In summary, the Semantic Web and related applications have received significant attention recently. The number, functionality and scope of the Semantic Web applications keep growing. Consequently, they result in the generation of more and more data in the Semantic Web language.

3.2. *The OWL 2 ontology language and the resource description framework (RDF)*

RDF is a language for describing things on the Semantic Web. Web resources are expressed as RDF triples or statements. Each triple has a subject, a predicate and an object. A triple represents a directed relationship between its subject and its object. A Uniform Resource Identifier (URI) reference within an RDF graph (an RDF URI reference) is a Unicode string that is used to identify a name or a resource over a network (typically the World Wide Web). Literals are used to represent values such as numbers or dates. In this work, we restrict literals to strings. The input to our

system is a small RDF graph which is the output of another system (e.g. a frequent subgraph miner [44, 45]). We assume that each graph is represented by a set of vertices and edges. Further details about RDF graph input into the NLGG system will be presented in Sec. 4.1.

Heath and Bizer [46] divide RDF triples into two categories. RDF links are used to express relationships between two resources. An RDF link consists of three URI references corresponding to the subject, predicate and object in the triple. The predicate URI refers to the type of relationship between two related resources. An example of RDF links is the triple "http://www.co-ode.org/roberts/family-tree.owl#john_william_folland http://www.co-ode.org/roberts/family-tree.owl#hasParent http://www.co-ode.org/roberts/family-tree.owl#m136" which is illustrated in Fig. 10. The "http://www.co-ode.org/roberts/family-tree.owl#john_william_folland", "http://www.co-ode.org/roberts/family-tree.owl#hasParent", and "http://www.co-ode.org/roberts/family-tree.owl#m136" are URI. RDF literals describe the properties of resources (e.g. name, date of birth of a person). The subject and predicate in an RDF literal are URIs, but the object is represented by a literal. An example of an RDF literal is "http://www.co-ode.org/roberts/family-tree.owl#john_william_folland http://www.co-ode.org/roberts/family-tree.owl#hasFamilyName "Folland"" where "http://www.co-ode.org/roberts/family-tree.owl#john_william_folland" and "http://www.co-ode.org/roberts/family-tree.owl#hasFamilyName" are URIs and "Folland" is a string.

An ontology is an explicit specification of conceptualization [47] which contains a description about concepts and their relationships. Ontology languages are formal languages used to construct ontologies. The OWL 2 Web Ontology Language is an ontology language for the Semantic Web. According to W3C [48], an OWL 2 ontology contains information about concepts, properties and types of relationships between objects in a specific domain. Any OWL 2 Ontology can be viewed as an RDF graph and vice versa. Classes define concepts. Individuals are instances of classes. Individuals represent resources/entities on the Semantic Web. Data values or datatype properties correspond to individuals' attributes or characteristics. Properties or Object properties refer to relationships between individuals. For simplicity, from now on we will use the word "*relationship*" to refer to an object property and "*attribute*" to refer to a data type property in an ontology. We use $P(x, y)$ to denote a direct relationship between x and y, where x, y, and P are correspondingly the subject, object and relationship type (or predicate). In OWL 2, a symmetric property P is a property such that if $P(x, y)$ is true then $P(y, x)$ also holds true. If two properties *P1* and *P2* are inverse then *P1(x, y)* is true iff *P2(y,x)* is also true.

The RDF/XML syntax [49] is standardized by the W3C and is widely used to publish Linked Data on the Web. The primary syntax for OWL 2 is RDF/XML. While RDF/XML provides a standard, other syntaxes may also be used. Alternative RDF specifications include, but are not limited to, Turtle [50], XML serialization [51], the Manchester syntax [52], and the functional-style syntax [51]. These can be easily changed into RDF/XML format by using tools like Jena (jena.apache.org).

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.co-ode.org/roberts/family-tree.owl#"
    xml:base="http://www.co-ode.org/roberts/family-tree.owl"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <owl:NamedIndividual rdf:about="http://.../family-tree.owl#john_william_folland">
            <rdf:type rdf:resource="http://.../family-tree.owl#Person"/>
            <hasFamilyName>Folland</hasFamilyName>
            <hasParent rdf:resource="http://.../family-tree.owl#m136"/>
      </owl:NamedIndividual>
</rdf:RDF>
```

Fig. 10. Example of RDF/XML onology.

The RDF/XML syntax is described in detail as part of the W3C RDF Primer [53]. An example of RDF triples in RDF/XML format is given in Fig. 10. The first triple states that there is a thing, identified by the URI "http://www.co-ode.org/roberts/family-tree.owl#john_william_folland" of type Person. The second triple states that this thing has the family name "Folland". The third triple states that "john_william_folland" has a "hasParent" relationship with a thing which is identified by the URI "http://www.co-ode.org/roberts/family-tree.owlm136".

3.3. *Semantic reasoners*

Reasoning is the process of inferring new facts that are not explicitly stated in the ontology or knowledge base. As Marek Obitko pointed out in his dissertation [54], a few tasks required from a semantic reasoner (or reasoner) are as follows.

- Satisfiability of a concept — determine whether there is a contradiction in the concepts' definitions.
- Subsumption of concepts — determine whether concept C is a subclass of concept D.
- Check an individual — check whether the individual is an instance of a concept.
- Retrieval of individuals — find all individuals that are instances of a concept.
- Realization of an individual — find all concepts the individual belongs to, especially the most specific ones.
- Retrieval of relationships — find all relationships related to the individuals in the ontology.

There are a number of existing reasoners both proprietary and free software. Jena (jena.apache.org) is a free Semantic Framework for Java. Jena has its own

specialized reasoning engine that works with RDF and OWL data sources. Also it is possible to connect the description logic reasoner Pellet [55] directly to Jena to have more reasoning power.

In short, RDF is the basic fabric behind the Semantic Web. Web resources are represented as RDF triples. Each triple contains URIs and may contain a string. The arbitrary nature of URIs makes Semantic Web documents hard for non-expert users to understand. Since natural language is the most human-friendly form or representation, and there are more and more data available in RDF format, we believe a natural language generation system whose task is to verbalize a set of RDF triples is needed. The next section presents our proposed NLGG system, its input format, system architecture and generation algorithm.

4. The NLGG System

Figure 1 presented an example of the input and output of our Natural Language Generation from Graphs (NLGG) system. NLGG is a template-based NLG engine. It uses a pipeline architecture which consists of three processing stages: (1) model preparation and content determination, (2) document structuring, and (3) lexicalization, aggregation and realization. Input to NLGG is a directed graph, along with the corresponding lexical information, and output from NLGG is a piece of English text. This section presents information about the input, type of output text and the architecture of the NLGG system.

4.1. *Input to NLGG*

As mentioned in Sec. 3, in a RDF/XML ontology, a class defines a concept. An individual is an instance of a class and is equivalent to a Semantic Web resource/entity. An object property represents a type of connection between two individuals. A datatype property represents an attribute/characteristic of an individual. Each class, individual, object property or datatype property is represented by a unique Universal Resource Identifier (URI). Literals are strings in our system. We use the words relationship and attribute to refer to an object property and data type property in an ontology, respectively. For simplicity, we will use alias names to refer to URIs. We use *isa* to represent a class definition predicate (which is actually http://www.w3.org/1999/02/22-rdf-syntax-ns#type). For other URIs, we eliminate the name space and use only the fragment identifiers to those URIs. For example, we use only *hasChild* to refer to the URI http://www.co-ode.org/roberts/family-tree.owl#hasChild, *Person* to refer to http://www.co-ode.org/roberts/family-tree.owl#Person, etc.

General RDF graphs consist of triples where only individuals will be at the two ends of a relationship (except class definitions) or attribute edge. However, output from some systems, such as a frequent subgraph miner, is not always in that form. They can have classes connected by relationship edges or a class link to a string by an attribute edge. Figure 11 gives an example of such a graph. The graph was taken

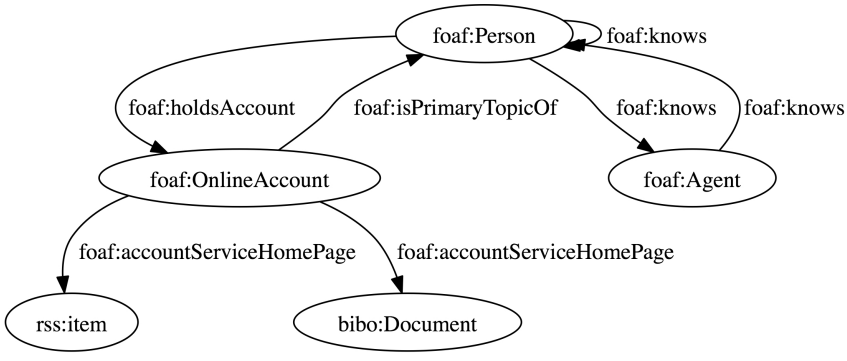

Fig. 11. Example output from a frequent subgraph miner applied to a large FOAF graph.

from the output of the frequent subgraph miner [44] on a semantic graph data set. We can see that all the nodes in the figure are general nodes. So to address the general nodes problem, the directed graph input to our natural language generator is a modified version of RDF graphs in which labels for vertices can be class URIs, individual URIs or strings. Labels for edges can either be relationship URIs or attribute URIs. Figure 11 illustrated a sample input graph to the system where "v" indicates a vertex, and "d" indicates a directed edge. Since any RDF graph can be easily converted into this form, the extension to the system input only enriches the coverage of the system.

Almost every NLG system needs syntactic and lexical information to be included in the knowledge base. An RDF URI reference is just a string. And as we pointed out in Sec. 2, automatically guessing semantic information from an RDF URI reference is problematic and thus is not feasible. Therefore, besides the small RDF graph, input to our system includes an ontology along with the template information for its classes, relationships and attributes. We try to keep the amount of manual work involved as small as possible. Therefore, the type of template input into our system is very simple. Further details about this are given below.

4.1.1. *Template information about classes, relationships and attributes*

For each class, we require the user to input information about its singular and plural forms. Existing morphology tools can generate plural forms of a single-word noun. But for a concept represented by multiple words like *BrotherInLaw* (refers to a brother in law), they fail to generate their correct plural forms. So NLGG requires plural forms of the concepts to be input as well. Table 1 illustrates template information for the *Brother*, *BrotherInLaw*, *Cousin*, *Daughter* and *DaughterInLaw* classes in the family ontology. *ClassID*, *singular*, and *plural* are, respectively, the class's URI, singular and plural form.

Because the NLGG system is designed to require minimal manual work, each template for a relationship/attribute consists of only the slots for basic syntax

Table 1. Example class templates for the family ontology.

ClassID	*ClassLabel*	*Singular*	*Plural*
http://.../family-tree.owl#Brother	Brother	brother	brothers
http://.../family-tree.owl#BrotherInLaw	Brother In Law	brother in law	brothers in law
http://.../family-tree.owl#Cousin	Cousin	cousin	cousins
http://.../family-tree.owl#Daughter	Daughter	daughter	daughters
http://.../family-tree.owl#DaughterInLaw	Daughter In Law	daughter in law	daughters in law

components in a sentence. For each relationship (object property), we require information about the following.

- verb
- priority
- object
- complement
- voice (passive/neutral)

Table 2 shows some example relationship templates, where *#subject* represents the subject in the triple, and *#object* represents the object in the triple. The *object* in the relationship template should be a noun or a noun-phrase. A complement is whatever comes after an object. We define a special string *null* to represent the absence of an object/complement in a relationship template. Priority gives system engineers the ability to order output facts in the same category. This is helpful when the system engineer might want specific types of information be output before others. Considering the family ontology, we might want information about blood and husband/wife relationships to come before information about in-law or other relationships. In this case, we set the priority of relationships like *hasParent*, *hasChild*, and *isWifeOf* higher than those of *hasAuntInLaw* and *isUncleOf*. Table 2 gives some of our relationship templates for the family ontology where smaller values mean higher priorities. Example 2 in Appendix A presents an example of how priorities help organize the content of the generated text. There are four relationships in the input graph. Information about in-law relationships (*hasBrotherInLaw* and *hasAuntInLaw*) comes before information about family relations (*isWifeOf* and *hasParent*). Without

Table 2. Example relationship templates from the family ontology.

ObjectPropertyID	*Priority*	*Verb*	*Object*	Complement	Voice
http://.../family-tree.owl#hasAuntInLaw	3	be	#subject's aunt	in law	passive
http://.../family-tree.owl#hasBrother	1	be	#subject's brother	null	passive
http://.../family-tree.owl#hasBrotherInLaw	3	be	#subject's brother	in law	passive
http://.../family-tree.owl#hasChild	1	be	#subject's child	null	passive
http://.../family-tree.owl#hasParent	1	be	#subject's parent	null	passive
http://.../family-tree.owl#isUncleInLawOf	3	be	#object's uncle	in law	neutral
http://.../family-tree.owl#isUncleOf	2	be	#object's uncle	null	neutral
http://.../family-tree.owl#isWifeOf	1	be	#object's wife	null	neutral

the use of priority, information about relationships will be output in that order. However, looking at the NLGG output text, since *hasParent* and *isWifeOf* have higher priorities, the text associated with them has been output before text about in-law relationships. In general, more important information should be given higher priority (smaller value). Facts in the same category (class definition, attribute or relationship information) will be organized according to relationship/attribute priorities. Further details will be given in the Document Structuring and Aggregation section. Table 2 demonstrates template information for eight relationships in the family ontology where *ObjectPropertyID* refers to a relationship URI.

Template information for an attribute is similar to that of a relationship. However, the "voice" field is replaced by "isBoolean" which is used to represent whether this is a boolean attribute or not (true/false value). Table 3 shows some examples of attribute templates for the family ontology, where *DatatypePropertyID* corresponds to an attribute URI. The GH "GrayHat" ontology is extracted from a defense-related project about people in a military-controlled area, their attributes and relationships (more detail is given in Sec. 5 describing experimental results).

4.2. *Preprocessing*

Suppose the user of the system is doing frequent subgraph mining on a RDF graph taken from a social network. The graph consists of information about thousands of users and their activities, their relationships, etc. Traditional NLG approaches, like that of the NaturalOWL system, would possibly build a large ontology from that large graph, annotate and then take it as input to their system. Apart from the huge amount of manual work, reasoning and querying on an ontology of such size is expensive and time consuming. As pointed out by Dentler *et al.* [56], the running time of an OWL DL and OWL 2 reasoner is exponential to the size of the input. So a larger ontology means much longer running time. In our experiments with the system, the Java Virtual Machine always ran out of memory even with a small ontology like the family ontology which contains 58 classes, 80 relationships, 9 attributes and 508 individuals and is equivalent to a set of 3,999 triples. It also ran out of memory when we use the GH ontology (with 3 classes, 11 attributes, 64 relationships and equivalent to $\sim 236,000$ triples).

As we have mentioned in Sec. 3, the size and scope of the Semantic Web graph data is growing day by day. Therefore, reasoning with the full ontology is not feasible. To overcome this, we reduced the size of the ontology input into our system by eliminating individuals' information from it. In other words, the ontology input to NLGG does not contain information about individuals but rather can be called an ontology skeleton in the sense that it only contains a list of classes, object properties and data type properties. Since the number of classes, relationships and attributes is limited, a lightweight ontology avoids the large-graph problem. For the family ontology, the size of the ontology skeleton is only 1/7 that of the original one (51 KB as opposed to 351 KB). Also the number of classes, relationships and attributes is only

Table 3. Example attribute templates from the family and GH ("GrayHat") ontologies.

DatatypePropertyID	*Priority*	*Verb*	*Object*	*Complement*	*Bool*
http://.../family-tree.owl#alsoKnownAs	1	be	null	also known as #object	False
http://.../family-tree.owl#formerlyKnownAs	1	be	null	formerly known as #object	False
http://.../family-tree.owl#hasBirthYear	1	be	null	born in #object	False
http://.../family-tree.owl#hasDeathYear	1	be	null	died in	False
http://.../family-tree.owl#hasFamilyName	1	have	family name: #object	null	False
http://.../ontology/GH/Name	1	be	null	called #object	False
http://.../ontology/GH/TimeOfEventStart	1	start	null	at #object	False
http://.../ontology/GH/isDetained	2	be	null	detained	True
http://.../ontology/GH/isSource	2	be	source	null	True

1/3 that of individuals ($58 + 80 + 9 = 147$ as opposed to 508), not to mention the fact that annotating an individual would take more time than a class/relationship/attribute (each individual has his/her own attributes and relationships).

We assume that the original large RDF graph is in RDF/XML format. If it is not, a tool like Jena can be used for the conversion. The RDF/XML ontology is then passed to a Java program for individuals' information removal. So the output of the preprocessing step is a skeleton ontology that contains only a list of classes, relationships and attributes. After this we need lexical information to be manually added into the NLGG knowledge base. This information is domain-dependent and represents the only manual work required for the NLG system to work. Information about these templates has already been given above.

4.3. *The NLGG architecture*

NLGG uses a pipeline architecture that generates natural language in three main stages: (1) model preparation and content determination, (2) document structuring, (3) realization, aggregation and lexicalization. Figure 12 illustrates the NLGG system architecture. Though aggregation is supposed to be one of the main tasks in the

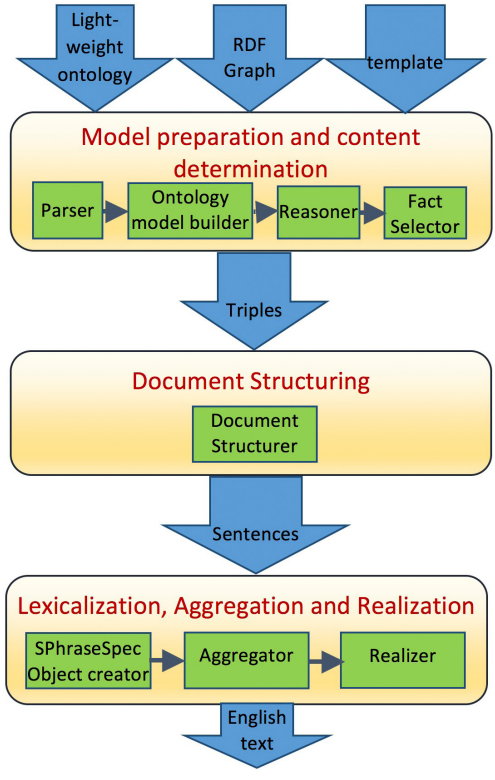

Fig. 12. The NLGG system architecture.

third stage, we do have some aggregation in the Document Structuring stage. The following sections will explain this in more detail.

4.3.1. *Model preparation and content determination*

NLGG's content determination task is not to decide what information should be communicated in the text. Instead, it is to decide in which way the information should be output to the user. The model preparation stage builds an ontology model at run time and adds in individuals' information to do the reasoning. As illustrated in Fig. 12, inputs to the system at this stage are: the lightweight ontology, an RDF graph and template information. Also the module can be decomposed further into six parts that are responsible for the following tasks.

(1) Read in the template information
(2) Load in the ontology model
(3) Parse information in the input graph to get a list of individuals, their attributes and relationships
(4) Add information in the graph to the ontology model
(5) Use Pellet reasoner to infer new information
(6) Select appropriate facts and remove redundancy

We use the Jena Semantic Framework (jena.apache.org) to load the ontology model from a file. Information about templates, individuals, attributes and relationships is parsed and stored as runtime dictionaries in the system.

As mentioned in the previous section, a reasoner's main tasks are to check the ontology consistency and infer new information from the existing logic. We use the Pellet reasoner [55] in our system. But any reasoner that understands Description Logic (DL) [57] and has an available Java API can also work well. We do not need to take the running time of the reasoners into account because for a small ontology, there should be little difference in their performance.

A reasoner like Pellet works only with facts related to individuals. It can only infer triples attached to individuals. However, as mentioned earlier, the NLGG knowledge base is only a lightweight ontology that contains no such knowledge. Therefore, the input RDF needs to be loaded into the ontology model at run time. Loading individuals' information at run time and removing it later without updating the ontology brings two main benefits. First, it makes reasoning possible. Second, it helps keep the size of the knowledge base small.

Inferring new information with a reasoner helps generate more natural text. Graph 1 of Example 1 in Appendix A gives an example of such capabilities. Instead of generating two sentences: "*A has child B. C has parent A*", the NLGG outputs "B and C are A's children". The *hasChild* and *hasParent* are represented as *inverse* relationships in the ontology. So "*C hasParent A*" and "*A hasChild C*" are two equivalent triples. Pellet understands this and thus is capable of inferring "*A hasChild C*" as a new fact. However, Pellet exhaustively generates all possible triples from the

Table 4. An exhaustive set of output triples from Pellet for the input graph in Fig. 1.

{subject, predicate, object}
{A, isa, Thing}
{B, isa, Thing}
{C, isa, Thing}
{A, isa, Person}
{B, isa, Person }
{C, isa, Person }
{A hasChild B}
{A hasChild C}
{B hasParent A}
{B hasParent A}

given logic. So both "*C hasParent A*" and "*A hasChild C*" are included in Pellet output. Table 4 gives a complete list of Pellet's output triples for the graph in Fig. 1. Since *Person* is a subclass of *Thing*. Triples like "*A isa Thing*" are also included in Pellet's output. Therefore, NLGG needs *Fact Selector* to choose "*A hasChild C*" and remove "*C hasParent A*" from the model. *FactSelector*'s selection criteria are as follows.

(1) Only consider statements whose subjects and objects have already existed as subject-object or object-subject pairs in the input graph. For example if the input RDF graph contains two statements "*A hasParent B*" and "*A isSiblingOf C*" and from that Pellet infers a new statement "*C hasParent B*", then that statement will be dropped by the *Fact Selector* because the input graph has no triple for *B* and *C*.
(2) Keep the set of class definition statements fixed. For example, if the input graph has "*A isa Person*" then keep only that triple and remove other *isa* triples like "*A isa Thing*" from the set of generated triples.
(3) For two inverse relationships, select the one that has higher priority (smaller value). If they have the same priority value, select the one that has higher alphabetic order. For example, in the family ontology, "*hasChild*" and "*hasParent*" have the same priority. Therefore, for graph 1 of Example 1 in Appendix A, "*A hasChild C*" was selected over "*C hasParent A*" according to alphabetic order.
(4) For symmetric relationships (e.g. if "*A isSiblingOf B*" then we also have "*B isSiblingOf A*"): keep only the existing statement and remove the newly inferred one.
(5) Remove new relationships inferred from rules (if any). Our objective is to keep the content of the input RDF graph intact. Using rules to replace "*A isSiblingOf B*" and "*A isa Male*" with "*A isBrotherOf B*" might make the output text look better. However, we should not replace "*A hasFather B*" and "*A hasHusband C*" with "*C hasFatherInLaw B*" because the replacement will result in the loss of

information (*A hasHusband C* and *hasFather B*). It is hard to decide when the inference engine should remove the triples in the left hand side of a rule, and when it should not. What is more, as we have discussed in Sec. 4.1, output from a frequent subgraph miner often contains general class nodes. There are also a number of other systems whose task is to mine concepts or motif patterns from a large semantic graph. Most of the time, the input RDF graph is used to express a general pattern. Inferring new information or further specializing the pattern based on inferences from the reasoner may change the intended generality or meaning of the pattern. For those two reasons, we will not make use of rules in the generation process.

A list of triples generated by *Fact Selector* will be passed as input to the document structuring and aggregation module. For graph 1 of Example 1 in Appendix A, the set of triples is given in Table 5.

4.3.2. *Document structuring*

As mentioned in Sec. 2, since NLGG has no notion of a focus point in the small input RDF graph, text planning strategies based on distance from the object being described cannot be applied. Instead, the system organizes information according to types. We group facts into three categories: class definitions, attribute information and relationship information; and organize them in that order. Also, information in the same group will be ordered according to priorities (which are included in the knowledge base). Class definitions are sentences that are derived from *isa* triples such as "*A isa Person*", "*C isa Document*", etc., where *A* and *C* are individuals, and *Person* and *Document* are concepts in the ontology.

Attribute information includes sentences that are derived from RDF triples in which the predicates are attribute URIs. "*Jack isSource "true"*", "*Olivia Gender "FEMALE"* ", or "*A age "20"*" are some examples of attribute triples where *Jack*, *Olivia* and *A* are individuals. Relationship information consists of sentences that are derived from RDF triples whose predicates are relationship URIs. "*A hasChild B*" and "*A knows B*", where *A* and *B* are two individuals, are examples of relationship triples. NLGG's document structuring strategy first outputs class definitions, then attribute information, and lastly relationship information.

Table 5. Output of the model preparation and content determination module for the input graph in Fig. 1.

{subject, predicate, object}
{A, isa, Person}
{B, isa, Person }
{C, isa, Person }
{A hasChild B}
{A hasChild C}

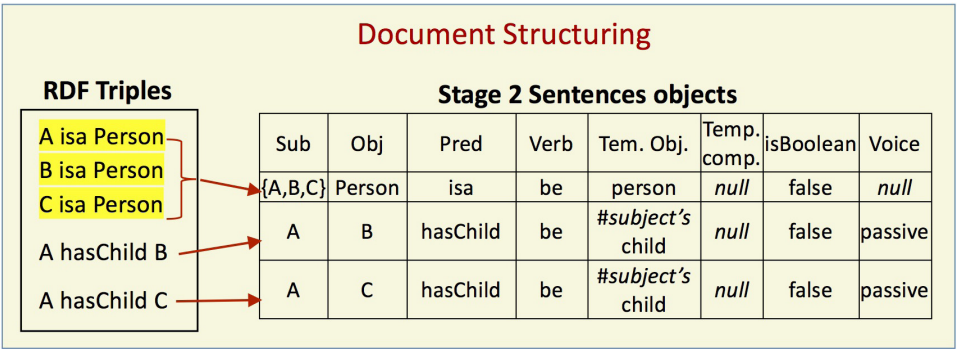

Fig. 13. NLGG document structuring for the RDF graph in Fig. 1.

At this stage, NLGG treats each sentence as a compound triple. A compound triple is a triple whose subject can consist of multiple individuals. Each sentence corresponds to a *Sentence* object. Each Sentence object has a list of subjects, a Boolean attribute representing whether the corresponding predicate is a Boolean property, and six other String objects for verb, predicate, object, template object, template complement and the voice. Output of the Document Structuring module is an ordered list of compound Sentences. The only aggregation job we do at this level is to wrap sentences that have the same object and predicate. In the input graph in Fig. 1 we have three sentences that are derived from three triples "*A isa Person*" and "*B isa Person*" and "*C isa Person*". When the aggregation rule is executed, they will be wrapped into just one sentence as presented in Fig. 13. An exhaustive list of compound sentences output from the module for the input graph in Fig. 1 is presented in the right hand table of Fig. 13. Each sentence object is represented as a row in the table. Sentence objects for class definitions always have the default *null* voice.

4.3.3. *Lexicalization, aggregation and realization*

We use simpleNLG [24] to realize a set of existing Sentences. Each Sentence is represented by an *SPhraseSpec* object in simpleNLG. Elements of that Sentence will be assigned to an appropriate component in that SPhraseSpec object depending on the sentence's voice. Figure 14 illustrates how the mapping works.

First, we replace possible *#subject* or *#object* in the Sentence's template slots with the Sentence's subject(s) or object, respectively. For a *neutral* or *null* voice Sentence, the Sentence's subject, verb, template object, and template complement will be assigned to its SPhraseSpec's subject, verb, object, and complement, respectively. The first Sentence output from the Document structuring in Fig. 14 is a *null* voice Sentence. The first row in the middle table in Fig. 14 presents the SPhraseSpec object corresponding to that Sentence. Here, {A,B,C} (the Sentence's subject) has been mapped to the SPhraseSpec's subject, the plural noun corresponding to the class Person (people) has been assigned to the SPhraseSpec's object.

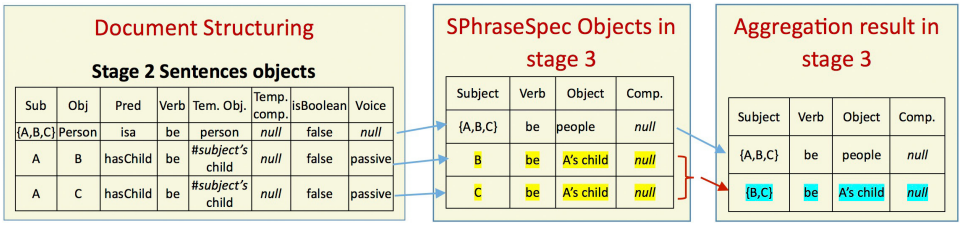

Fig. 14. NLGG lexicalization and aggregation for the RDF graph in Fig. 1.

The verb and template complement of the Sentence, which are "be" and "*null*", have been respectively assigned to the SPhraseSpec's verb and complement. The only difference between a negative voice Sentence's SPhraseSpec object construction and that of the other is that instead of mapping the Sentence's subject to the SPhraseSpec's subject, we assign the Sentence's template object to the SPhraseSpec's subject. The other mappings remain the same. We can see that for the second Sentence object in the leftmost table, the *#subject* has been replaced by A (the Sentence's subject) in its corresponding SPhraseSpec object in the middle table.

After getting a list of simpleNLG's SPhraseSpec objects, we aggregate them. We wrap sentences that have the same subject and verb but different objects into a compound sentence. We also wrap sentences that have the same verb and object but different subjects. Figure 14 demonstrates how these work. The second and third SPhraseSpec objects in the middle table have the same verb and object, and thus have been aggregated into just one SPhraseSpec object, which is represented as the second row in the rightmost table in Fig. 14.

A shows 12 of the 43 examples of text generated from the NLGG system. We can see that NLGG uses "*There is/are*" sentences to give identification to general nodes. General nodes of the same class are defined in one sentence. The number of "*there is/are*" sentences is equal to the number of different classes that have general nodes in the input graph. In graph 6, there are three general nodes that belong to three different classes. So the number of "*there is/are*" sentences added into the NLGG system is three.

5. Experimental Results

We constructed two experiments to evaluate NLGG. First, we evaluate how domain-dependent knowledge helps improve the quality of NLGG's generated text, and how NLGG works with ontologies in different domains. Second, we compare the effort required to make NLGG work with a new ontology to the effort for an existing NLG system. Details about the two experiments are presented below.

5.1. *Experiment 2: NLGG vs the simple verbalizer*

Our main approach to evaluate NLGG is to compare NLGG's output texts with those of a Simple Verbalizer. The Simple Verbalizer generation engine only uses some

simple rules to generate English texts. We observe that NLGG consistently outperforms the Simple Verbalizer. However, when the number of general nodes in the input graph increases, the NLGG's generation strategies reveal some problems. Also the system's capacity in generating coherent text is still limited. In this section, we present information about the Simple Verbalizer, its generation approach, the comparison results between the two systems and some discussion about NLGG limitations.

5.1.1. *The simple verbalizer*

The Simple Verbalizer is a simple rule-based NLG engine. It does not attempt to guess syntactic or lexical information from RDF triples. Instead, it uses some simple rules to verbalize them. Those rules are given below.

(1) Each triple will be output as one sentence.
(2) Place a period at the end of each sentence.
(3) The order of the output sentences is the same as that of the input RDF triples. In other words, the Simple Verbalizer reads in the input graph line by line, verbalizes each triple it encounters separately, and combines the output.
(4) For each RDF triple, its subject, predicate, and object are verbalized separately and combined in that order to form a sentence. So essentially, Simple Verbalizer treats an RDF triple as three ordered special strings that respectively correspond to its three components (subject, predicate and object). A special string can be a double quoted string or a URI. The Simple Verbalizer uses the same set of rules to verbalize each string and then combines the result. For example, for the triple "*A hasChild B*", Simple Verbalizer will verbalize "*A*", "*hasChild*", and "*B*" separately. "*A*" is verbalized as "*A*", "*B*" is verbalized as "*B*", and "*hasChild*" is verbalized as "*has child*". The results are then combined. A sentence "*A has child B*" is produced as the output of Simple Verbalizer for the triple "*A hasChild B*". The rules for verbalizing a special string are given below.

(a) For a double-quoted string, remove the double quote. For example, the string "*Armed Assault*" in the input graph in Example 6 of Appendix A will be output as *Armed Assault*.
(b) For a URI, eliminate the namespace in the URI. So for a URI like http://www.co-ode.org/roberts/family-tree.owl#hasParent, the http://www.co-ode.org/roberts/family-tree.owl# will be removed. Simple Verbalizer verbalizes only *hasParent* (the *fragment identifier*).
(c) For a *fragment identifier* that contains only uppercase letters such as *PERSON_RESPONSIBLE* in Example 6 of Appendix A, no changes are made. That means the URI http://eecs.wsu.edu/~ndong/ontology/GH/PERSON_RESPONSIBLE will be verbalized as *PERSON_RESPONSIBLE*.

(d) For other fragment identifiers: (1) replace underscores with spaces, (2) add spaces between a lowercase letter and an uppercase letter/digit and vice versa; or between an uppercase letter and a digit and vice versa. According to these, http://eecs.wsu.edu/∼ndong/ontology/GH/Person will be verbalized as *person*. http://eecs.wsu.edu/∼ndong/ontology/GH/EventActivityType will be verbalized as *event activity type*.

Refer to Appendix A for more examples of output texts from the Simple Verbalizer.

5.1.2. *Experiment data*

We evaluated NLGG on four different ontologies in different domains as described below. The data statistics are summarized in Table 6.

Table 6. Experiment data statistics.

Ontologies	No. of classes	No. of relationships	No. of attributes	No. of extracted graphs
FOAF	22	40	21	10
Family	53	80	9	5
University	43	25	7	10
GH	3	64	11	18

(1) The Friend of a Friend (FOAF) ontology from the FOAF project (www.foaf-project.org). It consists of 22 class definitions, 21 attributes and 40 relationships about people, their links and the things that they create and do.
(2) A family ontology downloaded from http://www.cs.man.ac.uk/∼ stevensr/ontology/family.rdf.owl. It contains 53 class definitions, 9 attributes and 80 different relationships about a family relationship hierarchy and related concepts.
(3) A university ontology from the SWAT projects (swat.cse.lehigh.edu/projects/lubm). It has 43 class definitions, 7 attributes and 25 different relationships about university people, their links and interaction.
(4) A real-world ontology, the GH "GrayHat" ontology, which is extracted from a defense-related project about people in a military-controlled area, their attributes and relationships. The ontology consists of 3 class definitions, 11 attributes and 64 different relationships.

From these ontologies, we manually created 43 small RDF graphs with varying complexity (10 from the FOAF, 5 from the family, 10 from the university, and 18 from the GH). We place no restriction on the number of RDF triples of the input RDF graph. The original ontologies were preprocessed to remove individuals' information. A program was used to extract the lists of classes, relationships and attributes from those ontologies. Twelve draft template files were created, manually edited, and then passed in as input to the system. We ran NLGG and the Simple Verbalizer on the 43 small RDF graphs that were created previously. Output texts were written to files and then manually compared.

5.1.3. *Experimental results and discussion*

5.1.3.1. Experiment 2.1

We processed the 43 small RDF graphs extracted from the four ontologies with both NLGG and the Simple Verbalizer. Output texts were written to files and manually compared. Appendix A gives the outputs from the Simple Verbalizer and NLGG for 12 of the 43 tested graphs. We presented the results in random order to two users, along with the input graphs, and asked them to select which text they prefer. We evaluated the results and observe that the users consistently prefer the NLGG text over the Simple Verbalizer in all the 43 input graphs. Also by manual analysis of the generated texts by the two systems, we draw the following observations.

(1) The texts generated by NLGG are more coherent, clearer and less stilted than those of the Simple Verbalizer.
(2) Simple Verbalizer cannot aggregate sentences while the NLGG can. NLGG can aggregate multiple similar class definitions into one sentence. Therefore, as illustrated in Example 1 of A, instead of saying that "*A is a person. B is a person. C is a person*" as the Simple Verbalizer did. NLGG says "*A, B and C are people*". In addition, NLGG can also aggregate facts associated with the same subject into one sentence. Looking at Example 3 of Appendix A, rather than using two sentences "*A publications C. A publications D*", NLGG aggregates them into "*A publishes D and C*". Clearly, NLGG aggregation makes the text briefer, clearer and more natural.
(3) NLGG document structuring makes the text look better. Example 4 of Appendix A illustrates such capabilities. Output text from the Simple Verbalizer flows in the same order as the input RDF graphs. The Simple Verbalizer outputs two sentences that contain information for the same subject A ("*A is a university*" and "*A affiliated organization of C*") far away from each other, one at the beginning and the other at the end of the output text. NLGG, on the other hand, is capable of organizing the output facts, placing those two sentences adjacent to each other, and then aggregating them to generate the sentence: "*A is an university and is affiliated with C*". We may notice a problem with the article generation by NLGG in this case. We will discuss this issue later. But for now, clearly, NLGG's capabilities in organizing and aggregating sentences make the texts shorter and clearer.
(4) NLGG can differentiate between different general class nodes while the Simple Verbalizer cannot. This makes the Simple Verbalizer generate texts failing to render the connections between objects in the graph, and thus makes the output texts look incoherent. Example 7 of Appendix A illustrates this issue. The input graph has a class node *Person* connected to a class node *Group* through the *PERSON_FOUNDER* relationship. And *Emma* is a member of that *Group*. Simple Verbalizer generates two sentences for those connections as "*Person PERSON_FOUNDER group. Emma PERSON_MEMBER group*". Clearly, there

is no notion of the connection between the group founded by some person and the group in which Emma has joined. Simple Verbalizer has failed to render the connection that was included in the input graph. NLGG, on the other hand, has successfully preserved such connections in its generated text by giving identification to general class nodes. It identified the person as *Person1* and the group as *Group1*. And the two triples were output as *"Person1 is the founder of Group1. Emma is a member of Group1.*" In general, NLGG can handle general class nodes while the Simple Verbalizer fails. It would be better if NLGG generated text like "*Person1 is the founder of a group. Emma is a member of that group.*" However, there are two main reasons why we do not support the generation of that kind of text. First, NLGG's document structuring strategy is object-oriented. That means facts about one individual might not concentrate to one part of the output text. In other words, there is no guarantee that the two sentences like "*Person1 is the founder of a group*" and "*Emma is a member of that group*" will be adjacent to each other in the output text. Therefore, using the relative pronoun is risky. Second, if we use "*a group*" to refer to a general *Group* node, then we might lose the connection between nodes when there are more than one general *Group* nodes in the graph. Similarly, if we have one general *Group* node, one general *Person* node, one general *Event* node, using a relative pronoun (like "*that group*", "*that person*", or "*that event*") might make the output text less coherent or even confusing. In general, such a generation approach only works with some specific cases. We do not place any restriction on the number of general nodes in the input RDF graph. Therefore, NLGG does not support the generation of texts that use relative pronouns to identify general nodes.

(5) Text generated by the NLGG is shorter than that of the Simple Verbalizer most of the time. NLGG generated texts are longer than that of the Simple Verbalizer only when there are general nodes in the graphs. That is due to the additional text generated by NLGG for identifying the general class nodes. Those sentences are necessary to preserve the connections to/from those nodes.

5.1.3.2. Experiment 2.2

We performed a second experiment with the 43 RDF graphs to test the effectiveness of the semantic reasoner in the generation process. We remove the reasoning module from the NLGG. We call the NLGG system without semantic reasoning: Simple NLGG. We pass the 43 RDF graphs with corresponding template information into Simple NLGG and let it generate natural language. Results are written into files. We observe that there are ten cases in which the generated texts by NLGG are different from those of the Simple NLGG. The input and generated texts by NLGG for seven of these ten graphs can be found in Appendix A in Examples 1, 2, 5 and 9–12. We gave a user the texts generated by the two systems along with the input graphs and asked him which texts he prefers. The user prefers NLGG's generated texts for nine out of the total ten graphs. He indicated the main reason for his preference is that

NLGG uses a consistent way to express symmetric or inverse relationships. Therefore, it can aggregate information better than the Simple NLGG. One example has been referenced throughout this work is that instead of generating "*B is A's child. A is C's parent.*" as output by the Simple NLGG, the NLGG with the reasoning capability can generate "*B and C are A's children.*" And most of the time, that kind of aggregation helps generate briefer and clearer texts. In summary, reasoning does help NLGG generate better text.

5.1.4. *Discussion*

The rest of this section is dedicated to further discussion about the NLGG capabilities and limitations in generating natural language. By analyzing the generated texts from NLGG, we observe the following.

(1) NLGG cannot generate sentences with connecting words, referring expressions or pronouns. As we discussed earlier when comparing NLGG with NaturalOWL, there are two main reasons for this problem. First, NLGG does not have a focus point. Thus, the text planning in NLGG cannot start from this object, and then expand further or explain further about that object. The graph typically contains information about one object or more objects, and then describes the attributes of each object, and then the relationships among objects. Changing the subjects so frequently makes it hard for NLGG to make use of relative pronouns or connecting words. Second, the more domain knowledge or semantic and syntactic information input to the system, the more power it has in generating natural text. Domain knowledge includes the knowledge that helps a NLG engine understand things like: "Person" refers to a person and we should use *he* to replace a person whose gender is male, etc. We try to avoid adding too much domain information into the system. We also try to keep the template information input into the system as simple as possible. So we consider this as a tradeoff between the quality of the generated text and the amount of handcrafted information input into the system.
(2) As previously pointed out, NLGG sometimes generates incorrect articles for a noun, e.g. Example 4 in Appendix A. This is because NLGG uses only simple rules to generate an article for a noun. It generates the article *an* whenever it encounters a noun begin with a vowel. So basically, it fails irregular cases like *university* as illustrated in the example.
(3) Sometimes the text dedicated to the general nodes' identification in NLGG is too much compared with the number of actual sentences generated from the triples. It would be better if NLGG can flexibly use relative clauses to avoid those kinds of general node identification sentences. For example, in Example 8 in Appendix A, the ideal generated text might be something like (1): "A person targets a Police Operation event" or (2): "A person targets an event which is a Police Operation event". Such expressions require the system to understand that

EventActivityType is a characteristic attribute and that we can use the "*Police Operation*" string to identify the general event node. This is considered domain dependent knowledge, and we do not want to require that kind of the knowledge be given to the system. So NLGG cannot generate sentences like: "A person targets an event which is a Police Operation", which requires the use of a relative clause. It looks simple with a graph with only two edges connecting to two general nodes. And the general nodes do not have any attribute edges connected to them. But it is a challenge when the number of general nodes in the graph increases and with the presence of attribute edges connecting to those general nodes. One example of this kind of complex graph is given in Example 6 in Appendix A. In general, when and where to use relative clauses is a challenge, especially when the number of general nodes increases.

5.2. *Experiment 2: NLGG vs NaturalOWL*

As pointed out in the related work section, NaturalOWL is the most complete NLG system that is closed to ours. In this experiment, we compare the quality of the text generated as well as the effort spent to make NLGG and NaturalOWL work for the M-PIRO ontology (which is contained in the NaturalOWL1.1 package and can be downloaded from http://protegewiki.stanford.edu/wiki/NaturalOWL_1.1). Our goal in this experiment is to validate our hypothesis that NLGG can generate acceptable text in a much more economical way compared with other NLG systems.

M-PIRO is an ontology for museum exhibits. It consists of 57 classes, 31 object properties, 6 data type properties and 343 individuals. We extract the ontology skeleton and create the domain dependent knowledge required by NLGG for M-PIRO. The new ontology skeleton size is 34 KB compared with the original one of 166 KB. Our templates size is reduced significantly from 889 KB to 10.3 KB. We do not know the time that was needed by domain experts to create domain knowledge for NaturalOWL, but it took us around one day to create templates for NLGG. We do believe bigger size means larger effort, not to mention the fact that NaturalOWL requires much more complex information than NLGG as discussed in previous sections. Hence, from the numbers, we claim that NLGG is cheaper and more portable than existing systems, namely NaturalOWL.

We manually extract 10 subgraphs from M-PIRO for 10 exhibits and let the two systems run for those inputs. Table 7 illustrates the generated text from the two systems for *exhibit5*. Refer to Appendix B for a sample of the results. As we can see in the table, NLGG text lacks referring expressions and hence makes it less coherent. We asked two people who are not domain experts to evaluate the results. They both prefer NaturalOWL's generated text but still approve that NLGG's text is acceptable. This is still consistent with our hypothesis that NLGG is a lightweight NLG system which can generate acceptable text with much less effort. There is always a trade off between high quality text and the amount of manual effort spent.

Table 7. NaturalOWL and NLGG text generated for exhibit5.

NaturalOWL	NLGG
This is a stater, created during the archaic period. It originates from Croton and it dates from between 530 and 510 B.C. It has a picture of a tripod on each side. A tripod is a vessel with three legs and it was the sacred symbol of god Apollo. This stater is made of silver and currently it is exhibited in the Numismatic Museum of Athens.	Exhibit 5 is a stater and is created during the archaic period. Exhibit 5 dates from between 530 and 510 B. C. It has a picture of a tripod on each side. A tripod is a vessel with three legs and it was the sacred symbol of god Apollo. Exhibit 5 is made of silver. Exhibit 5 originates from Croton. Exhibit 5 is currently exhibited in the Numismatic Museum of Athens.

We have presented our experimental results with the NLGG system. We performed experiments to see how NLGG compared with other existing systems, and how NLGG works with new ontologies in different domains. We summarized the system capabilities and the types of texts that can be generated from NLGG. In general, results have shown that NLGG can generate acceptable texts with reasonable cost. We also discussed some limitations with the systems. Those issues will be left for future work.

6. Conclusion

We have proposed a general NLG system to bridge the gap between Semantic Web representation and Natural Language so that people who are not familiar with RDF can also access the information and knowledge for various purposes. The use of a reasoner helps NLGG generate better and briefer text. Also, using an ontology skeleton instead of the full original one with individuals' information has helped NLGG to minimize the amount of manual work as well as reduce the system running time.

Four ontologies in different domains have been used to test the performance of the system. A Simple Verbalizer was used to compare the result. Experimental results showed that the NLGG consistently outperforms the Simple Verbalizer. The texts generated by NLGG are briefer, clearer and more coherent than those of the Simple Verbalizer.

Our objective is to minimize the amount of manual work involved in the knowledge base creation process. Also there is a tradeoff between the quality of the generated text and the amount of domain-dependent knowledge or lexical and syntactic information added into the system. Therefore, though there are still some limitations with the system, we consider the tradeoff acceptable. As this knowledge becomes more readily available, our NLGG approach can be expanded to take advantage of this information.

For future work, adding the referring expression generation module, connecting words and pronouns will help improve the quality of the generated text. However, it is tricky and might require the use of a different document structuring strategy. Also, testing the system with output from other real-world systems like graph-based

anomaly detection or results returned from a semantic web search agent would provide more insight into the performance of our approach.

Acknowledgments

The authors would like to thank the Vietnam Education Foundation for their support of Ms. Dong's graduate work.

References

[1] D. Galanis, G. Karakatsiotis, G. Lampouras and I. Androutsopoulos, An open–source natural language generator for OWL ontologies and its use in Protégé and second life, in *Association for Computational Linguistics*, 2009, pp. 17–20.

[2] X. Sun and C. Mellish, Domain independent sentence generation from RDF representations for the semantic web, in *Combined Workshop on Language–Enhanced Educational Technology and Development and Evaluation of Robust Spoken Dialogue Systems*, 2006.

[3] A. Cregan, R. Schwitter and T. Meyer, Sydney OWL syntax: Towards a controlled natural language syntax for OWL 1.1, in *OWL Experiences and Directions Workshop*, 2007.

[4] G. Hart, C. Dolbear and J. Goodwin, Lege feliciter: Using structured English to represent a topographic hydrology ontology, in *OWL Experiences and Directions Workshop*, 2007.

[5] K. Kaljurand and N. E. Fuchs, Verbalizing OWL in Attempto Controlled English, in *OWL Experiences and Directions Workshop*, 2007.

[6] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North and G. Woodhull, Graphviz and dynagraph–static and dynamic graph drawing tools, in *Graph Drawing Software*, 2003, pp. 127–148.

[7] E. Reiter and R. Dale, *Building Natural Language Generation Systems* (Cambridge University Press, 2000).

[8] J. Coch, Interactive generation and knowledge administration in MultiMeteo, in *Natural Language Generation*, 1998, pp. 300–303.

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

[10] S. Williams and E. Reiter, A corpus analysis of discourse relations for natural language generation, in *Corpus Linguistics*, 2003, pp. 899–908.

[11] S. Williams and E. Reiter, Generating readable texts for readers with low basic skills, in *European Workshop on Natural Language Generation*, 2005.

[12] F. de Rosis, F. Grasso and D. C. Berry, Refining instructional text generation after evaluation, *Artificial Intelligence in Medicine* **17**(1) (1999) 1–36.

[13] D. Huske-Krause, Suregen–2: A shell system for the generation of clinical documents, in *Association for Computational Linguistics*, 2003, pp. 215–218.

[14] F. Portet, E. Reiter, A. Gatt, J. Hunter, S. Sripada, Y. Freer and C. Sykes, Automatic generation of textual summaries from neonatal intensive care data, *Artificial Intelligence* **173**(7) (2009) 789–816.

[15] F. Benamara, Generating intensional answers in intelligent question answering systems, *Natural Language Generation*, 2004, pp. 11–20.

[16] N. Szilas, A computational model of an intelligent narrator for interactive narratives, *Applied Artificial Intelligence* **21**(8) (2007) 753–801.

[17] N. M. Sgouros, Dynamic generation, management and resolution of interactive plots, *Artificial Intelligence* **107**(1) (1999) 29–62.
[18] K. E. Thomas and S. Sripada, Atlas.txt: Linking georeferenced data to text for NLG, in *European Workshop on Natural Language Generation*, 2007.
[19] R. Power, Generating referring expressions with a unification grammar, in *Association for Computational Linguistics*, 1999, pp. 9–14.
[20] H. Somers, B. Black, J. Nivre, T. Lager, A. Multari, L. Gilardoni, J. Ellman and A. Rogers, Multilingual generation and summarization of job adverts: The tree project, in *Applied Natural Language Processing*, 1997, pp. 269–276.
[21] N. Colineau, C. Paris and K. Vander Linden, An evaluation of procedural instructional text, in *Natural Language Generation*, 2002, pp. 128–135.
[22] G.-J. Kruijff, E. Teich, J. Bateman, I. Kruijff-Korbayova, H. Skoumalova, S. Sharoff, L. Sokolova, T. Hartley, K. Staykova and J. Hana, Multilinguality in a text generation system for three slavic languages, in *Association for Computational Linguistics*, 2000, pp. 474–480.
[23] E. Reiter and R. Dale, Building applied natural language generation systems, *Natural Language Engineering* **3**(1) (1997) 57–87.
[24] A. Gatt and E. Reiter, SimpleNLG: A realisation engine for practical applications, in *Natural Language Generation*, 2009, pp. 90–93.
[25] E. Reiter, R. Robertson and L. M. Osman, Lessons from a failure: Generating tailored smoking cessation letters, *Artificial Intelligence* **144**(1) (2003) 41–58.
[26] D. Byron, A. Koller, J. Oberlander, L. Stoia and K. Striegnitz, Generating Instructions in Virtual Environments (GIVE): A challenge and evaluation testbed for NLG, in *Proceedings of the Workshop on Shared Tasks and Comparative Evaluation in Natural Language Generation*, 2007.
[27] E. Reiter and A. Belz, An investigation into the validity of some metrics for automatically evaluating natural language generation systems, *Computational Linguistics* **35**(4) (2009) 529–558.
[28] J. C. Lester and B. W. Porter, Developing and empirically evaluating robust explanation generators: The knight experiments, *Computational Linguistics* **23**(1) (1997) 65–101.
[29] C.-L. Yeh and C. Mellish, An empirical study on the generation of anaphora in chinese, *Computational Linguistics* **23**(1) (1997) 171–190.
[30] A. Belz, B. Bohnet, S. Mille, L. Wanner and M. White, The surface realisation task: Recent developments and future plans, in *Natural Language Generation*, 2012, p. 136.
[31] N. Bouayad–Agha, G. Casamayor, L. Wanner and C. Mellish, Content selection from semantic web data, in *Natural Language Generation*, 2012, pp. 146–149.
[32] R. Schwitter, K. Kaljurand, A. Cregan, C. Dolbear and G. Hart, A comparison of three controlled natural languages for OWL 1.1, in *OWL Experiences and Directions Workshop*, 2008, pp. 1–2.
[33] T. Berners-Lee, J. Hendler and O. Lassila, The semantic web, *Scientific American* **284**(5) (2001) 28–37.
[34] G. Kobilarov, T. Scott, Y. Raimond, S. Oliver, C. Sizemore, M. Smethurst, C. Bizer and R. Lee, Media meets semantic web: How the BBC uses DBpedia and linked data to make connections, in *The Semantic Web: Research and Applications*, 2009, pp. 723–737.
[35] K.-H. Cheung, E. Prudhommeaux, Y. Wang and S. Stephens, Semantic web for health care and life sciences: A review of the state of the art, *Briefings in Bioinformatics* **10**(2) (2009) 111–113.
[36] C. D. Pierce, D. Booth, C. Ogbuji, C. Deaton, E. Blackstone and D. Lenat, SemanticDB: A semantic web infrastructure for clinical research and quality reporting, *Current Bioinformatics* **7**(3) (2012) 267–277.

[37] C. Patel, S. Khan and K. Gomadam, Trialx: Using semantic technologies to match patients to relevant clinical trials based on their personal health records, in *International Semantic Web Conference*, 2009.
[38] B. Fazzinga, G. Gianforme, G. Gottlob and T. Lukasiewicz, Semantic web search based on ontological conjunctive queries, in *Web Semantics: Science, Services and Agents on the World Wide Web*, 2011.
[39] J. Gronski, Semantic web for search, in *Semantic Web Conference*, 2009, pp. 957–964.
[40] X. Jiang and A.-H. Tan, Learning and inferencing in user ontology for personalized semantic web search, *Information Sciences* **179**(16) (2009) 2794–2808.
[41] M. Grove and A. Schain, NASA expertise location service powered by semantic web technologies, in *W3C Semantic Web Case Studies and Use Cases*, 2008.
[42] J. Hendler, J. Holm, C. Musialek and G. Thomas, US government linked open data: Semantic.data.gov, *IEEE Intelligent Systems* **27**(3) (2012) 25–31.
[43] N. Shadbolt, K. O'Hara, T. Berners-Lee, N. Gibbins, H. Glaser and W. Hall, Linked open government data: Lessons from data.gov.uk, *IEEE Intelligent Systems* **27**(3) (2012) 16–24.
[44] C. Joslyn, S. Al-Saffar, D. Haglin and L. Holder, Combinatorial information theoretical measurement of the semantic significance of semantic graph motifs, in *SIGKDD Workshop on Mining Data Semantics*, 2011.
[45] V. Nebot and R. Berlanga, Finding association rules in semantic web data, *Knowledge-Based Systems* **25**(1) (2012) 51–62.
[46] T. Heath and C. Bizer, Linked data: Evolving the web into a global data space, *Synthesis Lectures on the Semantic Web: Theory and Technology* **1**(1) (2011) 1–136.
[47] T. R. Gruber, Toward principles for the design of ontologies used for knowledge sharing, *International Journal of Human Computer Studies* **43**(5) (1995) 907–928.
[48] O. W. G. W3C, OWL 2 web ontology language: Documents overview, *W3C Recommendation* **27** (2009) 1205–1214.
[49] D. Beckett and B. McBride, RDF/XML syntax specification (revised), *W3C Recommendation*, Vol. 10, 2004.
[50] D. Beckett and T. Berners-Lee, Turtle-terse RDF triple language, *W3C Team Submission*, Vol. 14, 2008.
[51] B. Motik, B. Parsia and P. F. Patel-Schneider, OWL 2 web ontology language XML serialization, *W3C Recommendation*, 2009.
[52] M. Horridge and P. F. Patel-Schneider, OWL 2 web ontology language manchester syntax, in *W3C Working Group Note*, 2009.
[53] F. Manola, E. Miller and B. McBride, RDF Primer, *W3C Recommendation* **10** (2004) 1–107.
[54] M. Obitko, Translations between ontologies in multi-agent systems, Czech Technical University, 2007.
[55] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur and Y. Katz, Pellet: A practical OWL-DL reasoner, *Web Semantics: Science, Services and Agents on the World Wide Web* **5**(2) (2007) 51–53.
[56] K. Dentler, R. Cornet, A. Ten Teije and N. de Keizer, Comparison of reasoners for large ontologies in the OWL 2 el profile, *Semantic Web* **2**(2) (2011) 71–87.
[57] I. Horrocks, OWL: A description logic based ontology language, *Principles and Practive of Constraint Programming*, pp. 5–8, 2005.

Appendix A. Selected Output Results from Simple Verbalizer and NLGG Systems

Example 1

v 1 http://www.co-ode.org/roberts/family-tree.owl#A
v 2 http://www.co-ode.org/roberts/family-tree.owl#Person
v 3 http://www.co-ode.org/roberts/family-tree.owl#B
v 4 http://www.co-ode.org/roberts/family-tree.owl#Person
v 5 http://www.co-ode.org/roberts/family-tree.owl#C
v 6 http://www.co-ode.org/roberts/family-tree.owl#Person
d 1 2 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 3 4 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 5 6 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 1 3 http://www.co-ode.org/roberts/family-tree.owl#hasChild
d 5 1 http://www.co-ode.org/roberts/family-tree.owl#hasParent

Simple Verbalizer	NLGG
A is a person. B is a person. C is a person. A has child B. C has parent A.	A, B and C are people. B and C are A's children.

Example 2

v 1 http://www.co-ode.org/roberts/family-tree.owl#A
v 2 http://www.co-ode.org/roberts/family-tree.owl#Person
v 3 http://www.co-ode.org/roberts/family-tree.owl#B
v 4 http://www.co-ode.org/roberts/family-tree.owl#Person
v 5 http://www.co-ode.org/roberts/family-tree.owl#C
v 6 http://www.co-ode.org/roberts/family-tree.owl#Person
v 7 http://www.co-ode.org/roberts/family-tree.owl#D
v 8 http://www.co-ode.org/roberts/family-tree.owl#Person
v 9 http://www.co-ode.org/roberts/family-tree.owl#E
v 10 http://www.co-ode.org/roberts/family-tree.owl#Person
d 1 2 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 3 4 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 5 6 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 7 8 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 9 10 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 3 5 http://www.co-ode.org/roberts/family-tree.owl#hasBrotherInLaw
d 3 7 http://www.co-ode.org/roberts/family-tree.owl#hasAuntInLaw
d 1 9 http://www.co-ode.org/roberts/family-tree.owl#hasParent
d 1 3 http://www.co-ode.org/roberts/family-tree.owl#isWifeOf

Simple Verbalizer	NLGG
A is a person. B is a person. C is a person. D is a person. B has brother in law C. B has aunt in law D. E is a person. A has parent E. A is wife of B.	A, B, C, D and E are people. A is E's child and B's wife. C is B's brother in law. D is B's aunt in law.

Example 3

v 1 http://xmlns.com/foaf/0.1#A
v 2 http://xmlns.com/foaf/0.1/Person
v 3 http://xmlns.com/foaf/0.1#B
v 4 http://xmlns.com/foaf/0.1/Person
v 5 http://xmlns.com/foaf/0.1#C
v 6 http://xmlns.com/foaf/0.1/Document
v 7 http://xmlns.com/foaf/0.1#D
v 8 http://xmlns.com/foaf/0.1/Document
d 1 2 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 3 4 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 5 6 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 7 8 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 1 3 http://xmlns.com/foaf/0.1/knows
d 1 5 http://xmlns.com/foaf/0.1/publications
d 1 7 http://xmlns.com/foaf/0.1/publications

Simple Verbalizer	NLGG
A is a person. B is a person. C is a document. D is a document. A knows B. A publications C. A publications D.	A and B are people. C and D are documents. A knows B. A publishes D and C.

Example 4

v 1 http://swat.cse.lehigh.edu/onto/univ-bench.owl#A
v 2 http://swat.cse.lehigh.edu/onto/univ-bench.owl#University
v 3 http://swat.cse.lehigh.edu/onto/univ-bench.owl#B
v 4 http://swat.cse.lehigh.edu/onto/univ-bench.owl#FullProfessor
v 5 http://swat.cse.lehigh.edu/onto/univ-bench.owl#C
v 6 http://swat.cse.lehigh.edu/onto/univ-bench.owl#ResearchGroup
d 1 2 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 3 4 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 5 6 http://www.w3.org/1999/02/22-rdf-syntax-ns#type

d 3 1 http://swat.cse.lehigh.edu/onto/univ-bench.owl#tenured
d 1 5 http://swat.cse.lehigh.edu/onto/univ-bench.owl#affiliatedOrganizationOf

Simple Verbalizer	NLGG
A is a university. B is a full professor. C is a research group. B tenured A. A affiliated organization of C.	C is a research group. B is a full professor. A is an university and is affiliated with C. B is tenured at A.

Example 5

v 1 http://swat.cse.lehigh.edu/onto/univ-bench.owl#College
v 2 http://swat.cse.lehigh.edu/onto/univ-bench.owl#University
v 3 http://swat.cse.lehigh.edu/onto/univ-bench.owl#A
v 4 http://swat.cse.lehigh.edu/onto/univ-bench.owl#Professor
v 5 http://swat.cse.lehigh.edu/onto/univ-bench.owl#B
v 6 http://swat.cse.lehigh.edu/onto/univ-bench.owl#University
v 7 http://swat.cse.lehigh.edu/onto/univ-bench.owl#C
v 8 http://swat.cse.lehigh.edu/onto/univ-bench.owl#D
v 9 http://swat.cse.lehigh.edu/onto/univ-bench.owl#E
v 10 http://swat.cse.lehigh.edu/onto/univ-bench.owl#ResearchGroup
d 3 4 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 5 6 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 7 1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 8 2 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 9 10 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 7 8 http://swat.cse.lehigh.edu/onto/univ-bench.owl#subOrganizationOf
d 3 7 http://swat.cse.lehigh.edu/onto/univ-bench.owl#headOf
d 3 5 http://swat.cse.lehigh.edu/onto/univ-bench.owl#doctoralDegreeFrom
d 3 9 http://swat.cse.lehigh.edu/onto/univ-bench.owl#memberOf

Simple Verbalizer	NLGG
A is a professor. B is a university. C is a college. D is a university. E is a research group. C sub organization of D. A head of C. A doctoral degree from B. A member of E.	E is a research group. A is a professor. C is a school. B and D are universities. A has a doctoral degree from B. A is the head of C and a member of E. C is a part of D.

Example 6

v 1 http://eecs.wsu.edu/∼ndong/ontology/GH/Person
v 2 “MALE”
v 3 http://eecs.wsu.edu/∼ndong/ontology/GH/Group

v 4 http://eecs.wsu.edu/∼ndong/ontology/GH/Event
v 5 "Armed Assault"
d 1 2 http://eecs.wsu.edu/∼ndong/ontology/GH/Gender
d 4 5 http://eecs.wsu.edu/∼ndong/ontology/GH/EventActivityType
d 1 4 http://eecs.wsu.edu/∼ndong/ontology/GH/PERSON_RESPONSIBLE
d 1 3 http://eecs.wsu.edu/∼ndong/ontology/GH/UNKNOWN

Simple Verbalizer	NLGG
Person gender MALE. Event event activity type Armed Assault. Person PERSON_RESPONSIBLE event. Person UNKNOWN group.	There is a Event, let's call it Event1. There is a Person, let's call it Person1. There is a Group, let's call it Group1. Event1 is an Armed Assault event. Person1 is a MALE, the responsible person of Event1 and an unknown to Group1.

Example 7

v 1 http://eecs.wsu.edu/∼ndong/ontology/GH/Group
v 2 http://eecs.wsu.edu/∼ndong/ontology/GH/Person
v 3 http://eecs.wsu.edu/∼ndong/ontology/GH#Emma
v 4 http://eecs.wsu.edu/∼ndong/ontology/GH#E1
v 5 http://eecs.wsu.edu/∼ndong/ontology/GH#Jayden
v 6 http://eecs.wsu.edu/∼ndong/ontology/GH/Person
v 7 http://eecs.wsu.edu/∼ndong/ontology/GH/Person
v 8 http://eecs.wsu.edu/∼ndong/ontology/GH/Event
d 3 6 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 4 8 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 5 7 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 3 4 http://eecs.wsu.edu/∼ndong/ontology/GH/PERSON_DEFENDANT
d 3 5 http://eecs.wsu.edu/∼ndong/ontology/GH/PERSON_FAMILY_CHILD
d 2 1 http://eecs.wsu.edu/∼ndong/ontology/GH/PERSON_FOUNDER
d 3 1 http://eecs.wsu.edu/∼ndong/ontology/GH/PERSON_MEMBER

Simple Verbalizer	NLGG
Emma is a person. E1 is a event. Jayden is a person. Emma PERSON_DEFENDANT E1. Emma PERSON_FAMILY_CHILD Jayden. Person PERSON_FOUNDER group. Emma PERSON_MEMBER group.	E1 is an event. There is a person, let's call it Person1. Jayden and Emma are people. There is a group, let's call it Group1. Emma is a child of Jayden and a defendant of E1. Person1 is the founder of Group1. Emma is a member of Group1.

Example 8

v 1 http://eecs.wsu.edu/∼ ndong/ontology/GH/Person
v 2 http://eecs.wsu.edu/∼ ndong/ontology/GH/Event
v 3 "Police Operation"
d 1 3 http://eecs.wsu.edu/∼ ndong/ontology/GH/EventActivityType
d 1 2 http://eecs.wsu.edu/∼ ndong/ontology/GH/PERSON_TARGETED

Simple Verbalizer	NLGG
Event event activity type Police Operation. Person PERSON_TARGETED event.	There is an event, let's call it Event1. There is a person, let's call it Person1. Event1 is a Police Operation event. Person1 targets Event1.

Example 9

v 1 http://xmlns.com/foaf/0.1#A
v 2 http://xmlns.com/foaf/0.1/Group
v 3 http://xmlns.com/foaf/0.1#B
v 4 http://xmlns.com/foaf/0.1/Person
v 5 http://xmlns.com/foaf/0.1#C
v 6 http://xmlns.com/foaf/0.1/Person
v 7 http://xmlns.com/foaf/0.1/Organization
d 1 2 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 3 4 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 5 6 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 3 1 http://xmlns.com/foaf/0.1/member
d 5 1 http://xmlns.com/foaf/0.1/member
d 3 5 http://xmlns.com/foaf/0.1/knows
d 5 7 http://xmlns.com/foaf/0.1/topic_interest

Simple Verbalizer	NLGG
A is a group. B is a person. C is a person. B member A. C member A. B knows C. C topic interest organization.	A is a group. C and B are people. There is an organization, let's call it Organization1. B knows C. C and B are members of A. Organization1 is the interested topic of C.

Example 10

v 1 http://eecs.wsu.edu/∼ndong/ontology/GH/Person
v 2 http://eecs.wsu.edu/∼ndong/ontology/GH#GH.N.TANGO.5264
v 3 "1000689718"

v 4 “ASNNFD”
v 5 http://eecs.wsu.edu/∼ndong/ontology/GH/Event
v 6 http://eecs.wsu.edu/∼ndong/ontology/GH#GH.N.TANGO.58421
v 7 “Police Operation”
v 8 “1115611”
d 2 1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 2 3 http://eecs.wsu.edu/∼ndong/ontology/GH/FamilyName
d 2 4 http://eecs.wsu.edu/∼ndong/ontology/GH/Ethnicity
d 6 5 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
d 6 7 http://eecs.wsu.edu/∼ndong/ontology/GH/EventActivityType
d 6 8 http://eecs.wsu.edu/∼ndong/ontology/GH/GH_EVT_ID
d 2 6 http://eecs.wsu.edu/∼ndong/ontology/GH/PERSON_TARGETED

Simple Verbalizer	NLGG
GH.N.TANGO.5264 is a person. GH.N.TANGO.5264 family name 1000689718. GH.N.TANGO.5264 ethnicity ASNNFD. GH.N.TANGO.58421 is a event. GH.N.TANGO.58421 event activity type police operation. GH.N.TANGO.58421 GH EVT ID 1115611. GH.N.TANGO.5264 PERSON TARGETED GH.N.TANGO.58421.	GH.N.TANGO.58421 is an event. GH.N.TANGO.5264 is a person. GH.N.TANGO.58421 is a Police Operation event. GH.N.TANGO.58421 has GH_EVT_ID 1115611. GH.N.TANGO.5264 is an ASNNFD. GH.N.TANGO.5264 has family name 1000689718. GH.N.TANGO.5264 targets GH.N.TANGO.58421.

Example 11

v 1 http://eecs.wsu.edu/∼ndong/ontology/GH/Event
v 2 http://eecs.wsu.edu/∼ndong/ontology/GH/Person
v 3 http://eecs.wsu.edu/∼ndong/ontology/GH/Group
v 4 http://eecs.wsu.edu/∼ndong/ontology/GH/Person
v 5 http://eecs.wsu.edu/∼ndong/ontology/GH/Group
d 2 3 http://eecs.wsu.edu/∼ndong/ontology/GH/PERSON_NETWORK_CONTACT
d 4 5 http://eecs.wsu.edu/∼ndong/ontology/GH/PERSON_LOGISTICS_CHIEF
d 4 5 http://eecs.wsu.edu/∼ndong/ontology/GH/PERSON_POLITICAL_MEMBER
d 2 4 http://eecs.wsu.edu/∼ndong/ontology/GH/PERSON_FAMILY_COUSIN
d 4 5 .../∼ndong/ontology/GH/PERSON_OPERATIONS_COMMANDER

Simple Verbalizer	NLGG
Person PERSON NETWORK CONTACT group. Person PERSON LOGISTICS CHIEF group. Person PERSON POLITICAL MEMBER group. Person PERSON FAMILY COUSIN person. Person PERSON OPERATIONS COMMANDER group.	There is an event, let's call it Event1. There are 2 people, let's call them Person1 and Person2. There are 2 groups, let's call them Group1 and Group2. Person2 is a cousin of Person1. Person1 is the logistics chief of Group1. Person2 has network contact with Group2. Person1 is the operations commander of Group1 and a political member of Group1.

Example 12

v 1 http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateStudent
v 2 http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateCourse
v 3 http://swat.cse.lehigh.edu/onto/univ-bench.owl#Course
v 4 http://swat.cse.lehigh.edu/onto/univ-bench.owl#Professor
d 1 2 http://swat.cse.lehigh.edu/onto/univ-bench.owl#takesCourse
d 1 3 http://swat.cse.lehigh.edu/onto/univ-bench.owl#teachingAssistantOf
d 1 4 http://swat.cse.lehigh.edu/onto/univ-bench.owl#advisor

Simple Verbalizer	NLGG
Graduate student takes course graduate course. Graduate student teaching assistant of course. Graduate student advisor professor.	There is a professor, let's call it Professor1. There is a course, let's call it Course1. There is a graduate student, let's call it GraduateStudent1. There is a graduate level course, lets call it GraduateCourse1. GraduateStudent1 is being advised by Professor1. GraduateStudent1 is taking GraduateCourse1. GraduateStudent1 is a teaching assistant for Course1.

Appendix B. Selected Output Results from NaturalOWL and the NLGG System

NaturalOWL	NLGG
This is a kouros that is 1.94 metres high. It dates from circa 530 B.C. It was secretly cut in two pieces and	Exhibit 2 is a kouros and is created during the archaic period. Exhibit 2 dates from circa 530 B. C. It was

(*Continued*)

NaturalOWL	NLGG
transported to Paris in 1937, before it was returned to Greece. This kouros is inscribed with "Stand and cry in front of the grave of dead Kroissos, who found death by Ares as he was fighting on the front". Currently this kouros is exhibited in the National Archaeological Museum of Athens.	secretly cut in two pieces and transported to Paris in 1937, before it was returned to Greece. Exhibit 2 is inscribed with "Stand and cry in front of the grave of dead Kroissos, who found death by Ares as he was fighting on the front". Exhibit 2 is currently exhibited in the National Archaeological Museum of Athens.
This is a statue that was sculpted by Polykleitos. It was created during the classical period and it is 2.12 metres high. (Actually, this is a Roman copy; we refer to the original). It dates from circa 440 B.C. and currently it is exhibited in the National Archaeological Museum of Napoli.	Exhibit 4 is a statue. (Actually, this is a Roman copy; we refer to the original). Exhibit 4 is created during the circa 440 B.C. Exhibit 4 dates from the classical period. Exhibit 4 is sculpted by Polykleitos. Exhibit 4 is currently exhibited in the National Archaeological Museum of Napoli.
This is a tetradrachm, created during the classical period. It originates from Attica and it dates from between 440 and 420 B.C. It is made of silver and currently it is exhibited in the Numismatic Museum of Athens.	Exhibit 6 is a tetradachm and is created during the classical period. Exhibit 6 dates from between 440 and 420 B. C. Exhibit 6 is made of silver. Exhibit 6 originates from Attica. Exhibit 6 is currently exhibited in the Numismatic Museum of Athens.
This is a drachma, created during the classical period. It originates from Attica and it dates from circa the 5th century B.C. It has an image of Athena crowned with a branch of olive, her tree, on one of its sides. One the other side, there is a picture of the goddess's owl. A drachma was enough for a "metic" (that is, a foreigner that stayed in Athens) to pay the "metic tax" each month. This drachma is made of silver and today it is exhibited in the Agora Museum of Athens.	Exhibit 8 is a drachma and is created during the classical period. Exhibit 8 dates from circa the 5 th century B. C. It has an image of Athena crowned with a branch of olive, her tree, on one of its sides. On the other side, there is a picture of the goddess's owl. A drachma was enough for a metic (that is, a foreigner that stayed in Athens) to pay the metic tax each month. Exhibit 8 is made of silver. Exhibit 8 originates from Attica. Exhibit 8 is currently exhibited in the Agora Museum of Athens.

(*Continued*)

NaturalOWL	NLGG
This is a marriage cauldron, created during the classical period. It was painted in the style of the Painter of Meidias and it was decorated with the red-figure technique. It originates from Attica and it dates from between 420 and 410 B.C. It depicts a bride wearing her "nymphides", meaning "bridal footwear". Today this marriage cauldron is exhibited in the National Archaeological Museum of Athens.	Exhibit 10 is a marriage cauldron and is created during the classical period. Exhibit 10 dates from between 420 and 410 B. C. Exhibit 10 depicts a bride wearing her nymphides, meaning bridal footwear. Exhibit 10 originates from Attica. Exhibit 10 is painted in the style of the Painter of Meidias. Exhibit 10 is decorated with the red-figure technique. Exhibit 10 is currently exhibited in the National Archaeological Museum of Athens.