Domain Independent Sentence Generation from RDF Representations for the Semantic Web

Xiantang Sun and **Chris Mellish** [1]

Abstract This paper presents a generic approach to sentence generation from RDF representations for the Semantic Web. We argue that RDF representations used in the Semantic Web contain rich implicit linguistic information, and that it may be feasible to achieve adequate domain independent generation from these representations using only generic knowledge sources.
Keywords: RDF graph, NLG, OWL

1 INTRODUCTION

The next generation of the web, the Semantic Web, integrates various distributed web resources from different domains using the Resource Description Framework (RDF) [15] syntax (mainly applied to instance data and ontologies). Unfortunately, these representations using the RDF syntax are not naturally human-readable like natural language. Natural Language Generation (NLG) techniques can help bridge the gap between the representations and natural languages in order that people who are not familiar with the RDF syntax can also access the information and knowledge for various purposes, e.g., educational applications that assist casual users to understand better about ontologies. Unfortunately, the cost of separately applying fine grained NLG techniques, which can generate text in high quality, to RDF representations for every domain would be very expensive and is thus not feasible. In fact, what most people require in this case is to basically understand the information or knowledge coded in the RDF syntax. Therefore, exploring an economic and generic way of generating acceptably readable text from RDF representations for any domain is necessary.

Facing this challenge, we find that the biggest obstacle in the way is that it is hard to achieve lexicalisation (e.g., lexicalising concepts in RDF representations and making lexical choices) without domain dependent lexicons [2]. Although there is some early research on generation from RDF instance data [2] [17], they are domain dependent and require hand-made lexicons. In this paper we present a generic approach to generation from RDF representations without hand-built lexicons. The approach is motivated by the results of our experiments to analyse a corpus of ontologies. Lexicalisation is the most significant part of our approach, and thus it is the emphasis of this paper.

Before moving to the details of our approach, we first discuss, in Section 2, RDF representations as an input to NLG, and review some other related work that also tackles NLG from RDF representations. In Section 3, we briefly present the results of our experiment to analyse a corpus of ontologies and summarise our conclusions about the linguistic nature of RDF representations. Then, our approach to generation from RDF representations without lexicons is presented informally, focusing on explaining our algorithm for lexicalisation (Section 4). We illustrate the algorithm by generating a sentence from an example RDF graph step by step. Finally, we present our initial conclusions and discuss our plans for future work in Section 5.

2 RDF REPRESENTATIONS AS INPUT TO NLG

Different NLG systems usually take different formats of input according to their generation strategies. Apart from conventional template-based approaches, which generate text using predefined patterns of discourse with blank spaces that need to be filled by users or applications, input to NLG systems could be roughly divided into two categories: tree-like notations, and non-hierarchical representations. A tree-like input notation is usually a semantic representation consisting of a predicate with its arguments. The predicate is lexicalised as the main verb of the sentence and the arguments are lexicalised as the complements of the main verb. In this kind of input, the semantics of sentences must be fully and explicitly represented using "tree" structures and descriptive constraints are often attached to the structures to provide control information. The advantage of using tree-like notations as inputs is that rich linguistic information and control information in the input enable NLG systems (e.g., KPML [1], and FUF/Surge [6]) to generate text of very high quality. However, as [11] discusses, dominance relationships between nodes in the semantics are often language-dependent and are not always preserved across languages, and the tree-like semantics assumption leads to simplifications which reduce the paraphrasing power of the generator. The other main type of input is non-hierarchical representations. The key idea of using non-hierarchical representations as input is that input to generators should represent semantics in a language-independent way and try to carry as little as possible linguistic information, so that the input is relatively "unbiased" and can leave sufficient room for generators to develop their paraphrasing power. Input of non-hierarchical representations can help overcome language boundaries because of the language independent nature of semantics. This is very useful for generating in multiple languages. A representative piece of work using non-hierarchical representations is [11]. In this work the inputs to the system are conceptual graphs [14]. They adopt an approximate matching strategy to map input conceptual graphs onto appropriate syntactic representations using mapping rules, which are declaratively specified in advance. The syntactic representations used in their work exploit the ideas of D-Tree Grammar [3]. Regardless of the exact input representations, NLG systems apart from template-based approaches require hand-built lexicons to achieve lexicalisation.

What kind of input representation is RDF? What kind of text do we need to generate from RDF representations? RDF was originally intended for situations in which information needs to be

1 University of Aberdeen, UK, email: {xsun,cmellish}@csd.abdn.ac.uk
2 Constructing lexicons is usually time-consuming and it is therefore one of the bottlenecks for cheaply applying NLG.

3 D-Tree Grammar originated from Tree-Adjoining Grammars [8].

processed by applications, rather than being only displayed to people. The information is logically represented using RDF syntax. In other words, RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning [15]. RDF representations belong to the class of non-hierarchical representations, because fundamentally they are *graphs*, rather than trees. Thus generation from RDF is similar to generation from semantic networks [13] or conceptual graphs. However we argue that RDF representations are more than logical representations, because they also contain rich linguistic information which can be used to greatly improve the quality of generation. Let's see an example. The group of RDF triples shown in figure 1 may be presented in natural language as:

LongridgeMerlot is a kind of Merlot. It has property locatedIn with value NewZealandRegion, property hasMaker with value Longridge, property hasSugar with value Dry, property hasFlavour with value moderate and property hasBody with value Light,

if these triples are seen as pure logical representations. However, a much better text could be made from these triples:

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

In the second case, firstly, we have recognized that *LongridgeMerlot* is the subject of the realised sentence because it is the subject of all the triples. Then we have figured out that *Merlot*, which is a noun, is the type of *LongridgeMerlot* by recognising *RDF:type*, thus the triple *(LongridgeMerlot, RDF:type, Merlot)* may be expressed as a short sentence consisting of a noun phrase (NP), *LongridgeMerlot*, and a verb phrase (VP), *is a kind of Merlot*, or a Noun phrase (NP), the *Merlot LongridgeMerlot* (please note that there may be more than one syntactic form for each triple, but here we only use one of them to demonstrate our idea). For the rest of the triples, *hasSugar Dry (VerbNoun Adj.), hasFlavor Moderate (VerbNoun Adj.), hasBody Light (VerbNoun Adj.), hasMaker Longridge (VerbNoun Noun)* and so on, we have turned them all to VPs, *has dry sugar, has moderate flavour, has light body, has maker Longridge* and so on. In order to get a more balanced sentence, the last three triples in the figure have been lexicalised as prepositional phrases (PP), *with dry sugar, with moderate flavour* and *with light body*. Then, we have aggregated the PPs to one PP, *with dry sugar, moderate flavour and light body.* Finally, we have integrated the syntactic segments to construct the sentence. In this case, we do not need any domain dependent lexicon to lexicalise the terms of the input. What we require here is the linguistic knowledge that can help us determine the combinations of words in these property and object names and reconstitute these "partially-lexicalised" fragments. Please note that the example of RDF graph here is a tree, but in practice RDF graphs may be in non-tree shapes.

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

Figure 1. A simple example of an RDF representation

3 EXPLORING THE LINGUISTIC NATURE OF RDF REPRESENTATIONS

In order to test our assumption that RDF representations in the scenario of the Semantic Web are not only logical representations but also contain rich linguistic information, we carried out an initial experiment (see [10] for more details about the results of our experiment). For this, we wrote a Java program using the Google API to help us look for online ontologies coded in OWL[4][16]. We obtained around five thousand links; however only some of these links were able to provide us with real ontology files, because firstly, Google limits its API not to be accessed more than one thousand times per day, and also because some of the links were not available. In total we collected 882 ontology files coded in OWL (111 Mb) as our corpus. In analysing these ontologies, we were interested in analysing the structure of two kinds of names: names of classes and names of properties. In order to detect the English words contained in these names, we used the WordNet [7] API to help us recognise English words occurring in these names. Each name is associated with a *pattern* recording its analysis as a sequence of parts of speech, where `Noun` etc. name standard parts of speech, `Than` and `Or` are used for particular closed class words that appeared in the corpus, `Un` names an unknown word and `C` names a capital letter not starting a recognised word (sequences of capital letters were not further analysed). So, for instance, *GreaterThanOrEqual* might be analysed as `AdjThanOrAdj` and *Student_IP* might be analysed as `NounCC`. Because there are no agreed rules for how to name concepts, people use various ways of giving names, which can be very long (*Muscle_Layer_of_Secondary_Duct_of_Left_Coagulating_Gland),* or even partially or entirely meaningless (*ICD10).*

In all, we found 3003 different patterns for class names (where 72% of names ended with nouns) and similar variation in property names (where 65% of names started with verbs). The most encouraging finding from this experiment was that only 14% of class names were totally unrecognised, and most property names (97%) could be at least partly recognised. Clearly, there is a considerable amount of linguistic material in these names, which implies that there is also a considerable amount of linguistic material in RDF representations.

Although RDF was originally intended for situations in which information needs to be processed by applications, rather than being only displayed to people, as we see from our experiment, RDF representations also contain significant rich linguistic information. This provides us with a good resource to help overcome the problem of achieving lexicalisation without lexicons. For example, *(id309 rdf:type RedWine) (id309 hasColour red)* can be realised as *"id309 is a type of RedWine and has a property, hasColour, with value red*", which has too much " RDF flavour". But by lexicalising the concepts using the hidden linguistic information, the generation can be "*id309 is a type of red wine, and it has red colour*." which is much more natural, though there is still a weak " RDF flavour" left.

[4] OWL is the most standard ontological language so far using the RDF syntax, and we assume that ideally RDF instance data should be all defined by ontologies in the scenario of the Semantic Web. Therefore, in theory results from the analysis of OWL ontologies may be representative of the nature of RDF representations which include both instance data and ontological data.

So far the examples used in this work are all from RDF instance data, which may be seen as "flat data". However, other representations using the RDF syntax could be much more complex than flat data, e.g., DAML/OIL [4] and OWL axioms. These representations have their own specialised terminologies, which, however, still obey the RDF syntax, and thus can be mapped onto RDF graphs. Although these representations can be viewed as sub-domains of RDF syntax and can be handled using the RDF syntax, the text generated from these representations purely based on the RDF syntax may be odd or even unreadable. So if we know about the syntaxes of these sub-domains, better output could be produced. For example, the three triples using the OWL syntax:

(redWine owl:Restriction _restriction1)

(_restriction1 owl:onProperty hasColour)

(_restrction1 owl:allValuesFrom red)

could be turned into text as,

redWine can only have red colour

which is better than the generation purely based on the RDF syntax:

redWine has a restriction _restriction1, which is on property hasColour and has all values from red

(in this example we have assumed that the implicit linguistic information hidden in the names can be understood). From this example, it is clear that understanding domain syntaxes is essential for generating naturally readable text. Therefore it will always be useful to be able to supplement default generation based on generic linguistic knowledge with special rules capturing arbitrary conventions for the domain. Currently, OWL is becoming the standard way to express ontologies in RDF. Our strategy is to explore the two extreme ends of RDF representations, RDF instance data and OWL data. At the current stage, we are applying our method of "free" lexicalisation to RDF instance data, but we will apply the idea to OWL in the near future.

4 "FREE LEXCALISATION" AND GENERATION FROM RDF REPRESENTATIONS

The input to our generator is a group of RDF triples representing a connected RDF graph and the output is a piece of text presenting the input. By default the generator assumes that the information carried by the input RDF graph is not too much for a single sentence (currently, we allow no more than 10 triples per input). Roughly speaking, our generation strategy is a microplanning focused strategy, which includes the conventional NLG processes lexicalisation and aggregation (but no referring expression generation). The main feature of this strategy is that our lexicalisation is "lexicon free" and takes place throughout the entire microplanning stage, interacting with aggregation. So, seen from this aspect, our strategy is a lexicalisation motivated strategy. In this section, we first introduce our methodology for lexicalisation and then informally present our generation algorithm.

4.1 "Free" lexicalisation

Before moving to our idea about how to achieve "free" lexicalisation by making use of the linguistic information hidden in RDF graphs, it is worth while introducing lexicalisation in theory. *Lexicalisation is a task which focuses on the problem of choosing which words should be used to pick out or describe particular domain concepts or entities* [12]. Generally there are two types of lexicalisation, *lexical realisation* and *true lexical choice* [3]. In the first case the lexicalisation process itself does little more than decide fine details of the realisation of lexemes, while in the second case significant decisions should be made about the choices between different lexemes representing essentially the same propositional content. In our work, we have to deal with both of these cases.

In our case of lexicalising RDF representations, we have to be able to do lexical realisation and true lexical choice depending on the implicit linguistic information that input RDF triples contain. We find some terms that contain adequate linguistic information can be directly lexicalised by simply separating the words in these terms and making up necessary auxiliary verbs or prepositions, e.g., the RDF triple, *(A, greater-than-or-equal-to, B)* can be realised to, *A is greater than or equal to B* by lexicalising the predicate to *is greater than or equal to* as a verb phrase (VP). But some terms only contain "key words", which must appear in the final lexicalised forms, e.g., the predicate in the triple *(A, Colour, red)*, so we have to compensate by adding appropriate words to make a complete and correct lexicalisation. Because the terms containing only "key words" leave us room to lexicalise them in quite different ways, we sometimes have to face the challenge of true lexical choice, e.g., *(A, Colour, red)* may be lexicalised as *A has colour red*, *A has red colour*, *A with red colour* or *red A*.

The discussion above only concerns the situation of lexicalising single RDF triples, however this is a very restricted case. When a group of RDF triples (an RDF graph) needs to be realised, lexicalisation becomes much more complicated because we have to consider not only how to lexicalise an isolated triple but also how to make appropriate lexical choice by considering the lexicalisation in the triple's context. To be precise, we argue that there are three key factors that influence lexical choice in our case:

1. *Consistency and potential for aggregation*. Lexicalisation of a triple should be consistent with the lexicalisations of the triples in its *neighbourhood* which is the group of triples sharing the same subject. This notion may be extended in other RDF representations, e.g., OWL. An RDF graph may be viewed as a group of connected neighbourhoods. Currently, we only consider the case that lexicalised triples with the same main verb or preposition are consistent and have good potential for future aggregation, e.g., in the lexicalisations of the triples, *(ID309 hasColour red) (ID309 hasTaste strong)*, we think *ID309 has red colour and strong taste* is more consistent than *ID309 with red colour has strong taste.*

2. *Contextual syntactic constraints*. In some cases, earlier lexicalised RDF triples in an RDF graph force other triples which are connected with them in the graph to be lexicalised in a way syntactically consistent with the earlier lexicalisation. For example, assume that (*ID309 hasProducer producer123*) and *(producer123 hasHistory 50_years)* need to be presented in a sentence.

After the first triple is realised as *ID309 has producer producer123*, the second triple has to be lexicalised as a relative clause rather a sentence so that the two lexicalisations can be integrated together to a sentence, i.e. as *ID309 has producer producer123, which has a history of 50 years*, To a large extent, the *contextual syntactic constraints* also need to take into account the co-occurrence patterns among the nodes in the input graphs, i.e. the relative clause is only allowed semantically because the object of the first triple is the same as the subject of the second.

3. *Balanced sentences.* Sometimes when input RDF graphs are large, the final generated sentences may contain a large NP as the main subject with a short VP, or vice versa, a short subject with a large VP of complex structure. How can we avoid these extreme cases and obtain relatively balanced final generation of sentences? This requires us to consider the *balance* factor when making a lexical choice. So far, we only have a simple algorithm to achieve the balance of sentences by calculating the length of each possible lexicalisation and ordering them in the most balanced way (see Section 4.2 for the details of the algorithm). Note that the notion of *balance* might vary among different kinds of RDF representations, e.g., RDF instance data and OWL data. Our current goal is to explore a generic way to calculate *balance*, which can be applied across RDF representations. If this proves to be unsuccessful, we will formulate strategies for RDF instance data and OWL separately.

In summary, the challenges to achieve automatic "free lexicalisation" are:

1. Be able to understand the implicit linguistic information in the input and use this resource to lexicalise single RDF triples in most cases (including understanding the RDF syntax), also dealing with the worst case when there is no embedded linguistic information in input triples. Our system's capability for dealing with the implicit linguistic resource is essential for us to meet this challenge. Our experiment gives us encouragement that this will be possible.

2. Make appropriate lexical choices to lexicalise RDF triples and make them consistent with their lexicalised neighbourhood triples.

3. Make appropriate lexical choices to lexicalise RDF triples without breaking their *contextual syntactic constraints*.

4. Be able to calculate the balance of generated sentences and produce relatively balanced generation.

For the first challenge, we decide to write rules based on the generic patterns discussed in Section 2 to map input RDF triples which have been pre-analysed using WordNet onto their lexicalised forms. The advantage here is that we map input directly to their syntactic forms without knowing any of the semantics hidden in the input (other NLG techniques, excluding template-based ones, have to know the semantics of the input). A rule consists of two parts, patterns on the left that can match RDF triples, and all possible syntactic forms on the right which correspond to the patterns on the left. Basically, there are three kinds of rules. The first kind is the rules which are mainly applied to RDF triples containing almost all required lexical information. Usually, this kind of rule can only map one RDF triple onto one syntactic form. A simplified rule is shown in Figure 2. Using this rule, for example, an RDF triple, *(A, is_Physical_Part_of, B)* can be lexicalised as, *LTAG(S(NP(A) AuxVerb(is) NP(a physical part of B)))*. The second kind of rule is rules which are mainly applied to the RDF triples containing only key lexical information. Usually, this kind of rule may produce multiple syntactic forms for one RDF triple. For instance, *(A, Colour, red)* may be lexicalised using rule2 as *LTAG(NP(NP* (NP(A) PP(PP(with) NP(red colour))))* or *LTAG(S(NP(A) VP(VP(has) NP(red colour))))* or *LTAG(NP(NP* (NP(A) WH VP(has) NP(red colour)))* (The * indicates where an adjunction may take place. The details of our syntactic representations, based on Tree Adjoining Grammar [8], are not shown). The third kind of rules are defined to handle special cases in the RDF syntax; these are not discussed here. So far we have implemented 31 rules for exploiting the implicit linguistic information and 11 rules for RDF syntax.

For the second challenge, currently we only view syntactic structures with the same main verb in the same neighbourhood as consistent structures. For the third challenge, we decide to use Lexicalised Tree-Adjoining Grammar (LTAG) [9] to represent our syntactic structures. In LTAG, every syntactic structure (elementary tree) explicitly points out what other syntactic structures are allowed to be adjoined with it and how. In other words, the *contextual syntactic constraints*, are naturally expressed in the LTAG trees when we constructing these trees. Other work has also used the TAG family of grammar formalisms because of this key advantage [5] [5] [11]. For the last challenge, how we improve our algorithm of achieving *balanced sentences* will depend on our future experiments.

Example

Rule1: *(A,[AuxiliaryVerb][Adj][Noun][Prep],B)→*

LTAG (S(NP(A)+[AuxiliaryVerb] +NP([Adj]+[Noun]+[Prep] +B)))

Rule2: *(A, [Noun] , [Adj])→ LTAG (NP(NP* (NP(A)+*

PP(PP('with')+[Adj]+[Noun])))) OR

LTAG (S(NP(A)+ VP(VP('has')+NP([Adj]+[Noun])))) OR

LTAG(NP (NP(NP(A)+WH+ VP(VP('has')+NP([Adj]+[Noun]))))*

Figure 2. Two examples of system rules

[5] The key advantage of TAG is that the extended domain of locality over which syntactic dependencies are stated and function argument structure is captured [8]. We will not present any further details about LTAG here because of space limitations. Knowledge of LTAG is not required to understand this paper.

4.2 Generation algorithm

In this section we present the generation algorithm using a simple example of RDF instance data. At the current stage we have only applied the algorithm to RDF instance data, though the algorithm itself is designed to work also with OWL data. We limit the number of input RDF triples (=10) because we believe an input RDF graph carrying too much information should not be presented in one sentence. So it is the users' responsibility to determine a sensible size of content as input to the generator. For generation from large RDF graphs, users have to decompose the large graphs into smaller sub-graphs of sensible sizes as input and the generator outputs single sentence for each sub-graph separately. In general, domain-dependent principles must be used to determine the content and organisation of multiple sentence output, and so we do not consider this step here. Currently, our algorithm requires the input RDF graphs to be rooted directed acyclic graphs, which are more like tree structures. But our future goal is to develop an improved algorithm that can tackle RDF graphs as directed graphs with the full generality of Nicolov's approach with conceptual graphs. At the current stage, an RDF graph can be tackled by our generator only if the graph can be transformed into a tree whose depth[6] is no more than three.

The following algorithm is a possible solution to lexicalising RDF graphs. The patterns obtained from our experiment (section3) only provide us with the limited power to lexicalise single triples in RDF graphs and we have to find a way to integrate them. Our idea is to assemble the triples in the same neighbourhood, lexicalise the neighbourhoods according to our preferences, and integrate these lexicalised neighbourhoods into a complete syntax tree, which can be easily turned to text. Therefore, our algorithm is a bottom-up solution. Below are the details of the algorithm.

The algorithm has three main steps, as follows:

Step 1: Transforming an input RDF graph into a tree by splitting nodes with multiple input edges and producing an initial lexicalisation.

1a: G0 is an input RDF graph, T=Tree(G0) is the transformed graph as a tree structure,

1b: N=Neighbourhood(T) is the set of neighbourhoods in T. L=LexNeighbourhood(T) is the set of lexicalised neighbourhoods, obtained by applying the rules to the triples in N.

Step 2: Constructing a balanced noun phrase for every neighbourhood in the tree.

2a: **for all** li ∈ L **do**

2b: **if** li contains more than 3 triples or li plus its sons contain more than 4 triples

2c: **then** PP_shift(li)

2d: **end if**

2e: **end for**

2f: **for all** li ∈ L **do**

2g: ai=Aggregate(li) //All lexicalised triples with the same verb and PP are aggregated. Any aggregated PP is shifted to stay with the subject of li. During the aggregation, related lexicalised triples are put together. So li is converted to a neighbourhood-level syntactic structure.

2h: **end for**

Step 3: Constructing a complete lexicalised syntax tree.

3a: s = Adjoining(A) // this process adjoins all neighbourhood-level syntactic structures, A, to a complete lexicalised syntax tree, s.

3b: Realising(s)

We now explain some of these steps in more detail. The transformation in step 1a is not detailed here but is illustrated in the progression from Figure 4 to Figure 5. Here we focus on steps 1b and 2 which are the key processes of the generation.

In step 1b, for every RDF triple t ∈ Tree(G), Rule(t) is the result of lexicalising t to a VP. By default every triple's property and object can be lexicalised as a VP, but sometimes it can be also lexicalised as a PP (the lexical choice depends on the following processes). If T is the tree resulting from step 1a, Neighbourhood(T) is the partition of the triples in T into those subsets which share the same subject node. Thus, Neighbourhood(T) = $\{n_i \mid i=1..m\}$, where m is the number of nodes in T that have outgoing triples, each n_i is the set of triples with the same given subject, and in terms of triples, $T = n_1 \cup n_2 \cup \ldots \cup n_m$. Then LexNeighbourhood(T), the lexicalised neighbourhoods of T, is defined by:

LexNeighbourhood(T)= {{Rule(t)|t∈ N}|N∈ Neighbourhood(T)}

A lexicalised neighbourhood l_i is a set of lexicalised triples which contains VPs. In every lexicalised neighbourhood, lexicalised triples that have the same verb are grouped as having potential for future aggregation. We use VPG to indicate a group of lexicalised triples (in a lexicalised neighbourhood) that have the same main verb, and VPG$_i$ to indicate the partition of l_i into subsets which have the same main verb. Now every lexicalised neighbourhood is ready to be turned into a NP plus relative clause by assigning the shared subject as the main subject of the clause and aggregating all lexicalised triples to get a main VP for the clause.

However, if a neighbourhood contains too many triples, this implies that an unbalanced relative clause with a small subject and a large aggregated VP may occur. Our solution is to re-lexicalise one or two triples (in the same VPG) that can also be turned into PPs and shift them to stay with the subject in order to achieve

[6] *The depth of a graph is the length of the longest path which does not visit a node more than once.* For example, *(A, P1,B1)(A, P2,B2)(B1,Q,C)* is a graph with depth 2, and *(A, P,B)(B,Q,C) (C,L,D) is a graph with depth 3.* Our algorithm generates NPs corresponding to neighbourhoods of the transformed graphs, so large depth will lead to multiply embedded NPs, which can lead to readability problems (even though they may be grammatically correct).

better balance. We believe too much information should not be shifted to the subject in order to avoid an over-large subject, so no more than two triples are allowed to be PP-shifted.

Now we introduce how PP-shift works. In the algorithm, PP_shift(l*i*), where l*i* is a lexicalised neighbourhood, is triggered if l*i* contains more than three lexicalised triples or its own triples plus the triples in its son neighbourhoods [7] number more than four triples. We argue that it is important to maintain a good trade-off between the increased balance of the generation from a neighbourhood by shifting PPs and the increased complexity of the structure of the generation after the shift of PPs. For example, assume a sentence with the syntactic structure, *NP1+VP(VP1+VP2+VP3)*, where all information carried by *VP2* can be PP-shifted and *VP3* can be partially PP-shifted. In this example, we think that NP(*NP1+PP2) + VP(VP1+VP3)*, obtained by PP-shifting the phrase *VP2*, will produce a better text than *NP(NP1+PP3)+VP(VP1+VP2+VP3)*, where *PP3* is from partially PP-shifting *VP3* and *VP3* is the rest after the shift, because the first case increases the balance and also reduces the complexity of the main VP of the sentence, while the second case increases the complexity of the entire structure though the balance is increased. We argue that if a neighbourhood, l*i*, has too many VPGs (we think >2 is too many), the benefit of increased balance from a PP-shift that can not reduce the number of VPGs is counteracted by the increased complexity, and thus PP-shift in this case is not recommended. In the algorithm, if l*i* contains more than 2 VPGs, PP_shift(l*i*) is allowed only if the shift can reduce the number of VPGs; if l*i* contains no more than 2 VPGs, PP-shifts may take place in either VPG (if both VPGs are available, the bigger one is chosen. In one VPG, PP-shift always takes one triple if there are not exactly two related available triples, e.g., *(id728 family_name Green)* and *(id728 given_name Peter)*, which contain the same string apart from their verbs if they have). This is how PP-shift works in the algorithm.

As its output, the algorithm computes a set of local syntactic structures A, which contains a structure a*i* for each neighbourhood n*i*, and these are then adjoined to produce the final answer.

Now let's see an example of generating a sentence from a simplified RDF graph in Figure 3 which is graphically shown in Figure 4. **(step1)** Firstly, the RDF graph is transformed into a graph with the tree structure shown in Figure 5. Using the rules in Figure 3, the triples are lexicalised as short clauses with internal syntactic structure, e.g.,

LTAG (S((NP(id728)+VP(" is a" +male))),

LTAG(NP(NP(NP(id728)+WH+VP(VP('has')+"a"+NP ([family name] + [Green])))))*

and so on (* is where adjunction may take place, but details of adjunction are not presented here). There are three neighbourhoods in the example: neighbourhood 1 including triples 1-6, neighbourhood 2 including triples 7 and 8, and neighbourhood 3 including triple 9. These give rise to three lexicalised neighbourhoods. **(step2)** In lexical neighbourhood 1 there are more than 3 triples and only 2 VPGs (a VPG for *is* and a VPG for *has)*. According to the algorithm, triple 2 and triple 3 in the VPG for *has* are chosen for PP-shift (because 1. the VPG of *has* is bigger than the VPG of *is* and 2. the two triples are related because they both contain the string *name*). Then all lexicalised triples in neighbourhood 1 are aggregated to give the syntactic representation of:

id728 with family name Green and given name Peter is a male, and has a homepage http://www.xyz.ac.uk, an assistant Amy, and an office _recource1.

(step3) All neighbourhood-level syntactic structures are adjoined,

id728 with family name Green and given name Peter, is a male and has a homepage http://www.xyz.ac.uk, an assistant Amy which has an address University of A, UK, and an *office _recource1 which has a telephone 12345 and an address University of A, UK.*

After some trivial operations like removing anonymous nodes, e.g., *_resource1*, the final text is

id728 with family name Green and given name Peter, is a male and has a homepage http://www.xyz.ac.uk, an assistant Amy which has address University of A ,UK, and an office which has a telephone 12345 and an address University of A, UK.

4.3 Discussion

Our generation algorithm can often generate adequately readable text from RDF instance data, and the quality of the output depends on how much linguistic information is carried by the input and what structure that the input graph has. In the extreme case that there is no linguistic information embedded in the input, we have default rules to produce text purely based RDF syntax. One drawback of the algorithm is the limited way in which related triples are detected (just looking for an occurrence of the same string). In addition, the parameters in the algorithm, e.g., currently, we set 2 as the maximum number of triples that a PP-shift allows, should be adjusted based on our future experiments. We have implemented this algorithm which works well with simple examples, but we have not yet done a formal evaluation because the quality of final output text depends on not only this algorithm but also other parts of our system, e.g., our developing algorithm that can transform an RDF graph into a tree. A formal evaluation will be done once we integrate all parts of our system.

5 CONCLUSION AND FUTURE WORK

In this paper our domain independent approach to sentence generation from RDF representations is presented. The main feature of our solution is that our lexicalisation is " free" and helps produce consistent and balanced sentence. So far we have applied our approach to RDF instance data, but we may face more difficulties in the generation from OWL, which is much more complicated than instance data. As our next task, we plan to do experiments on how real readers choose their preferred balanced sentences from the results of our generator, and will formulate a better strategy of *balance* based on the experiment results**.**

[7] We call a neighbourhood A, whose subject is an object of a triple in neighbourhood B, a son of neighbourhood B. This relation can also be applied among lexicalised neighbourhoods.

1. *(id728 rdf:type male)*
2. *(id728 family_name Green)*
3. *(id728 given_name Peter)*
4. *(id728 office _resource1)*
5. *(id728 assistant Amy)*
6. *(id728 homepage http://www.xyz.ac.uk)*
7. *(_recource1 telephone 12345)*
8. *(_recource1 address "University of A, UK")*
9. *(Amy address "University of A, UK")*

Rule1: *(A, [Noun1] , [Noun2])→ LTAG (NP(NP* (NP(A)+ PP(PP('with')+ [Noun1] +[Noun2])))) OR LTAG(NP(NP*(NP(A)+WH+ VP (VP ('has')+ A/AN+ NP ([Noun1] + [Noun2])))))*

Rule2: *(A,rdf:type ,B)→ LTAG (S((NP(B)+VP(" is a "+A)))*

Figure 3. Example input and rules

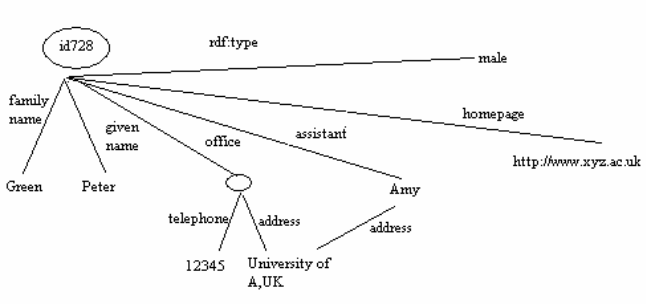

Figure 4. Initial input RDF graph before transformation

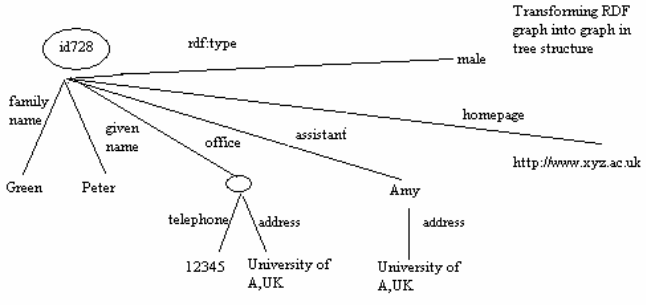

Figure 5. Input in tree structure after transformation

REFERENCES

[1] J. A. Bateman. Enabling technology for multilingual natural language generation: The KPML development environment. *Natural Language Engineering*,3:15-55,1997.

[2] K. Bontcheva and Y. Wilks.. Automatic report generation from ontologies: the MIAKT approach In *Nineth International Conference on Applications of Natural Language to Information Systems (NLDB'2004)*. Manchester, UK, 2004

[3] L. Cahill. Lexicalisation in applied NLG systems. Technical Report ITRI-99-04, Information Technology Research Institute (ITRI), University of Brighton, 1998. Available at http://www.itri.brighton.ac.uk/projects/rags.

[4] DAML Project. http://www.daml.org.

[5] B. Eddy, D. Bental and A. Cawsey. An algorithm for efficiently generating summary paragraphs using tree-adjoining grammar. In *Proceedings of the 8th European Workshop on Natural Language*. Toulouse, France, 2001.

[6] M. Elhadad, and J. Robin. An overview of SURGE: a reusable comprehensive syntactic realization component. Technical Report 96-03. Department of Computer Science, Ben Gurion University, Beer Sheva, Israel, 1996.

[7] C. Fellbaum. *WordNet: An Electronic Lexical Database.* MIT Press, Cambridge, Massachusetts, 1998.

[8] A. K. Joshi. The relevance of tree adjoining grammar to generation. In *Natural Language Generation: New results in Artificial Intelligence, Psychology and Linguistics-NATO Advanced Research Workshop*, pages 233-252, Nijmegen, The Netherlands, 1986.

[9] A. K. Joshi and Y. Schabes. Tree adjoining grammars and lexicalized grammars. In Maurice Nivat and Andreas Podelski, editors, *Definability and Recognizability of Sets of Trees*, 1991.

[10] C. Mellish and X. Sun. The Semantic Web as a *Linguistic* Resource: Opportunities for Natural Language Generation. Presented at the *Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, Cambridge, 2005.

[11] N. Nicolov, C. Mellish, and G. Richie. Approximate Generation from Non-Hierarchical Representations. In *Proceedings of the 8th International Workshop on Natural Language Generation*, Herstmonceux Castle, UK, 1996.

[12] E. Reiter and R. Dale. Building applied natural language generation systems. *Natural Language Engineering* 3, pages 57–88, 1997.

[13] R. Simmons and J. Slocum. Generating English discourse from semantic networks, *Comm. ACM* 15, 891- 905 (1972).

[14] J. F. Sowa. Conceptual graphs summary. In Timothy E. Nagle, Janice A. Nagle, Laurie L. Gerholz, and Peter W. Eklund, editors, *Conceptual structures: Current Research and Practice*, Ellis Horwood Series in Workshops, pages3-51. Ellis Horwood Limited, London, England, 1992.

[15] W3C. RDF vocabulary description language 1.0: RDF schema W3C proposed recommendation, In *World Wide Web Consortium*, 2003, at http://www.w3.org/TR/rdf-schema

[16] W3C. OWL Web Ontology Language Reference, In *World Wide Web Consortium*, 2004, at http://www.w3.org/TR/owl-ref/

[17] G. Wilcock and K. Jokinen. Generating Responses and Explanations from RDF/XML and DAML+OIL. In: *Knowledge and Reasoning in Practical Dialogue Systems, IJCAI-2003, Acapulco*, pages 58-63, 2003.