Generating Tailored Textual Summaries from Ontologies

Kalina Bontcheva*

Department of Computer Science,
University of Sheffield, Regent Court,
211 Portobello Street, Sheffield, UK
kalina@dcs.shef.ac.uk

Abstract. This paper presents the ONTOSUM system which uses Natural Language Generation (NLG) techniques to produce textual summaries from Semantic Web ontologies. The main contribution of this work is in showing how existing NLG tools can be adapted to Semantic Web ontologies, in a way which minimises the customisation effort while offering more diverse output than template-based ontology verbalisers. A novel dimension of this work is the focus on tailoring the summary formatting and length according to a device profile (e.g., mobile phone, Web browser). Another innovative idea is the use of ontology mapping for summary generation from different ontologies.

1 Introduction

The Semantic Web aims to add a machine tractable, re-purposeable[1] layer to compliment the existing web of natural language hypertext. In order to realise this vision, the creation of semantic annotation, the linking of web pages to ontologies, and the creation, evolution and interrelation of ontologies must become automatic or semi-automatic processes.

Natural Language Generation[2] (NLG) takes structured data in a knowledge base as input and produces natural language text, tailored to the presentational context and the target reader [8]. NLG techniques use and build models of the context and the user and use them to select appropriate presentation strategies. For example, deliver short summaries to the user's WAP phone or a longer multimodal text if the user is using their desktop.

In the context of Semantic Web or knowledge management, NLG can be applied to provide automated documentation of ontologies and knowledge bases.

* This work is partially supported by the EU-funded SEKT (http://sekt.semanticweb.org) and KnowledgeWeb (http://knowledgeweb.semanticweb.org) projects.

[1] Re-purposeable in this case meaning useful in a number of different applications, i.e. application-independent.

[2] For an in-depth introduction to NLG see [8].

A. Gómez-Pérez and J. Euzenat (Eds.): ESWC 2005, LNCS 3532, pp. 531–545, 2005.

Unlike human-written texts, an automatic approach will constantly keep the text up-to-date which is vitally important in the Semantic Web context where knowledge is dynamic and is updated frequently. The NLG approach also allows generation in multiple languages without the need for human or automatic translation (see [1]). This is an important problem firstly because textual documentation is more readable than the corresponding formal notations and thus helps users who are not knowledge engineers to understand and use ontologies. Secondly, a number of applications have now started using ontologies to encode and reason with internally, but this formal knowledge needs to be also expressed in natural language in order to produce reports, letters, etc. In other words, NLG can be used to present structured information in a user-friendly way.

There are several advantages to using NLG rather than using fixed templates where the query results are filled in:

- NLG can use different sentence structures depending on the number of query results, e.g., conjunction vs itemised list.
- depending on the user's profile of their interests, NLG can include different types of information – affiliations, email addresses, publication lists, indications on collaborations (derived from project information).
- given this variety of what information from the ontology can be included and how it can be presented, depending on its type and amount, writing templates will be unfeasible because there will be too many combinations to be covered.

This variation comes from the fact that it is expected that each user of the system will have a profile comprising of user supplied (or system derived) personal information (name, contact details, experience, projects worked on), plus information derived semi-automatically from the user's interaction with other applications. Therefore, there will be a need to tailor the generated presentations according to user's profile.

NLG systems that are specifically targeted towards Semantic Web ontologies have started to emerge only recently. For example, there are some general purpose ontology verbalisers for RDF and DAML+OIL [12] and OWL [11]. They are based on templates and follow closely the ontology constructs, e.g., *"This is a description of John Smith identified by http://...His given name is John..."* [11]. The advantages of Wilcock's approach [12, 11] is that it is fully automatic and does not require a lexicon. A more recent system which generates reports from RDF and DAML ontologies is MIAKT [3]. In contrast to Wilcock's approach, MIAKT [3] requires some manual input (lexicons and domain schemas), but on the other hand it generates more fluent reports, oriented towards end-users, not ontology builders. It also uses reasoning and the property hierarchy to avoid repetitions, enable more generic text schemas, and perform aggregation.

At the other end of the spectrum are sophisticated NLG systems such as TAILOR [7], which offer tailored output based on user/patient models. Systems like Wilcock's [11] and MIAKT [3] tend to adopt simpler approaches, exploring generalities in the domain ontology, because their goal is to lower the effort for

customising the system to new domains. Sophisticated systems, while offering more flexibility and expressiveness, are difficult to adapt by non-NLG experts. For example, experience in MIAKT showed that knowledge management and Semantic Web ontologies tend to evolve over time, so it is essential to have an easy-to-maintain NLG approach.

This work extends the MIAKT approach towards making it less domain dependent and easier to configure by non-NLG experts. A novel dimension is the focus on tailoring the summary formatting and length according to a device profile (e.g., mobile phone, Web browser). Another innovative idea is the use of ontology mapping for summary generation from different ontologies.

The paper is structured as follows. Section 2 introduces the ONTOSUM system and its architecture. Next Section 3 focuses on portability and customisation, including an extended example, using an existing Semantic Web ontology. The formatting and length tailoring algorithms are discussed in Section 4. The paper concludes with a discussion of future work.

2 System Architecture

Since ONTOSUM is designed to be part of interactive applications, it needs to *(i)* respond to user requests in *real-time*, i.e., avoid generation algorithms with associated high computational cost; and *(ii)* be *robust*, i.e., always produce a response. Consequently the system uses some efficient and well-established applied NLG techniques such as text schemas and a phrasal lexicon (see [8]).

The ONTOSUM system is implemented as a set of components in the GATE infrastructure [2], which provides an easy-to-use graphical development environment for NLP systems. In particular, we make use of its ontology support, which provides language-independent access to ontologies. The advantage of this approach is that our generator can handle RDF, DAML+OIL, and OWL, without any modifications, as it uses the format-independent GATE API, rather than format-specific ones.

Similar to other applied NLG systems (see [8]), ONTOSUM is implemented as pipeline system, i.e., the generation modules are executed sequentially.

Summary generation starts off by being given a set of statements (i.e., triples), in the form of RDF/OWL. Since there is some repetition, these triples are first pre-processed to remove already said facts. In addition to triples that have the same property and arguments, the system also removes triples involving inverse properties with the same arguments, as those of an already verbalised one. The information about inverse properties is provided by the ontology (if supported by the representation formalism).

Next is the summary structuring module, which orders the input statements in a coherent summary. This is done using discourse patterns, which are applied recursively and capitalise on the property hierarchy (see Section 3.2). This module also performs *semantic aggregation*, i.e., it joins together statements with the same property name and domain, so they are expressed within one sentence (see Section 3.3).

Finally, the generator transforms statements from the ontology into conceptual graphs [10] which are then verbalised by the HYLITE+ surface realiser [3]. The output is a textual summary (see Figure 3).

A similar approach was first implemented in a domain- and ontology-specific way in the MIAKT system [3]. In ONTOSUM we extended it towards portability and personalisation, i.e., lowering the cost of porting the generator from one ontology to another and generating summaries of a given length and format, dependent on the user target device. These issues are discussed in detail next.

3 Portability and User-Friendliness

Natural Language Generation (NLG) systems consist of two types of components: domain-dependent and domain-independent ones. Typically the text structuring component is domain-dependent, because every domain or application tends to have different conventions for what constitutes a coherent text. Another example domain-dependent module is the lexicon which maps concepts to their lexical items and grammatical information. Therefore, when an NLG system is adapted to a new domain or application, these components need to be modified.

In contrast, the surface realisation module, i.e., the module that generates the sentences, given their formal syntactic structure, is typically domain-independent and does not need to be adapted.

Therefore, this section will focus on the lexicalisation and summary structuring modules in ONTOSUM, while the HYLITE+ surface realiser will not be covered, as it is domain-independent (see [3]).

3.1 Lexicalisations of Concepts and Properties

The lexicalisations of concepts and properties in the ontology can be specified by the ontology engineer, be taken to be the same as concept names themselves, or added manually as part of the customisation process. For instance, the AKT ontology[3] provides `label` statements for some of its concepts and instances, which are found and imported in the lexicon automatically:

```
<daml:DatatypeProperty rdf:ID="has-email-address">
  <rdfs:label>has email address</rdfs:label>
  <rdfs:range rdf:resource="&xsd;string"/>
  <rdfs:isDefinedBy rdf:resource="&base;"/>
</daml:DatatypeProperty>
```

The generator is parameterised at run time by specifying which properties are to be used for building the lexicon, e.g., `label`, `name` (in the SWRC ontology[4]) (see the `propertyNames` parameter in Figure 2).

[3] http://www.aktors.org/ontology/
[4] http://ontoware.org/projects/swrc/

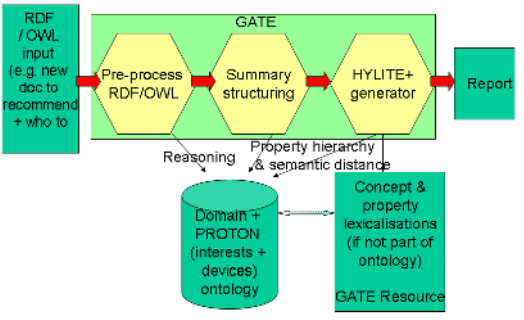

Fig. 1. ONTOSUM's Architecture

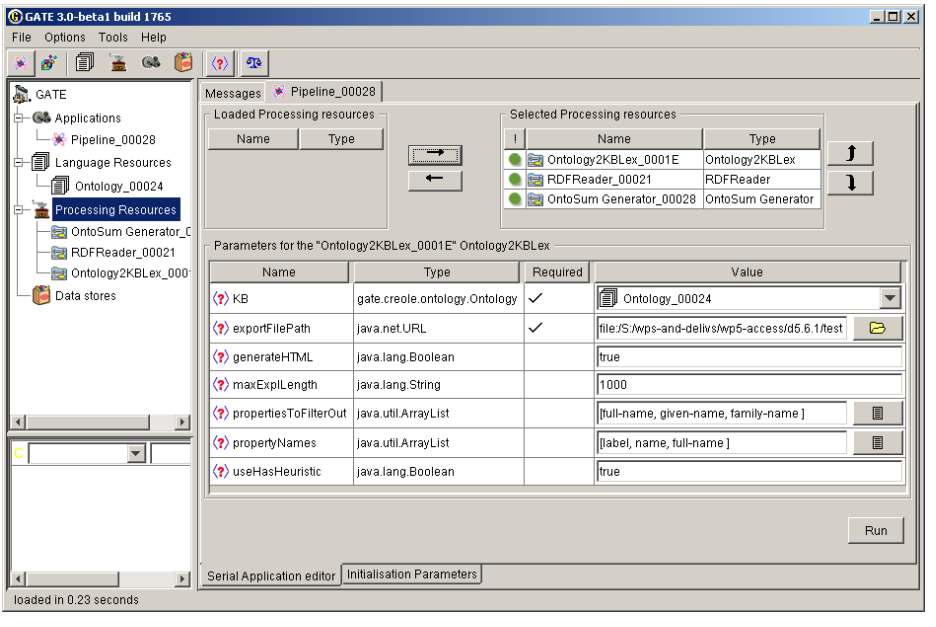

Fig. 2. Example ONTOSUM Configuration Parameters

The lexicon is thus generated automatically from the ontology. By default concepts are assumed to be lexicalised as nouns and properties as verbs. This is a rather strong simplification, but given that it is true in many cases, it does save the user the effort of having to specify these manually for the entire ontology. Instead, the user only needs to verify that the automatically assigned part of speech is correct and only change the exceptions. The lexical entries are in the format `<Concept-Name, Lexicalisation, GrammaticalFeatures>`. The grammatical features are a list of attribute-value pairs, e.g., `pos` – part-of-speech (noun, verb, adjective, etc.), `num` – number (singular or plural), `massnoun` – value is `true` if this is a mass noun, i.e., uncountable nouns like water. Some sample entries are shown below:

```
lex_entry_eng('System-Administrator',
              'system administrator',[fs(pos,noun),fs(num,sing)]).
lex_entry_eng('Generalised-Means-Of-Transport',
              'Generalised-Means-Of-Transport',[fs(pos,noun),fs(num,sing)]).
```

In this example, the lexicalisation of the first entry was taken from its `label` property, but the second entry did not have such a property, so the concept name was assigned as a lexicalisation. The user can then edit the name as they edit the part of speech and other grammatical information.

3.2 Summary Structuring

Discourse/text schemas, as introduced by [5], are script-like structures which represent discourse patterns. They can be applied recursively to generate coherent multisentential text satisfying a given, high-level communicative goal.[5] Each schema consists of rhetorical predicates (e.g. *comparison, constituency*) which encode communicative goals and structural relations in the text. Rhetorical predicates are also associated with a semantic function which selects appropriate statements from the ontology. In this way, by selecting and instantiating schemas, a text structuring component can produce coherent texts which satisfy given communicative goals.

In more concrete terms, when given a set of statements about a given concept/instance, discourse schemas are used to impose an order on them, such that the resulting summary is coherent. For the purposes of our system, a coherent summary is a summary where similar statements are grouped together.

The top-level schema for describing instances from the ontology is:

```
Describe-Instance ->
    Describe-Attributes,
    Describe-Part-Wholes,
    Describe-Active-Actions,
    Describe-Passive-Actions
```

where `Describe-Attributes`, etc. are recursive calls to other schemas. For example, the `Describe-Attributes` schema collects recursively all properties that are sub-properties of the `attribute-property` and involve the given instance:

```
Describe-Attributes ->
    [attribute(Instance, Attribute)],
    Describe-Attributes *
```

The schemas are independent of the concrete domain and rely only on a core set of 4 basic properties – `active-action`, `passive-action`, `attribute`, and `part-whole`. When a new ontology is connected to ONTOSUM, properties can be defined as a sub-property of one of these 4 generic ones and then ONTOSUM will be able to verbalise them without any modifications to the discourse

[5] For instance, `(definition AIRCRAFT CARRIER)` – generate text that defines aircraft as a type of carrier – (example D, [5–p.44]).

schemas. However, if more specialised treatment of some properties is required, it is possible to enhance the schema library with new patterns, that apply only to a specific property.

Since most ontologies do not have these 4 properties in their property hierarchy (e.g., AKT ontology[6], SWRC ontology[7]), we implemented a heuristic for recognising attributive properties as a way of lowering the adaptation effort. If this heuristic is enabled, the generator considers all properties with names starting with `has` as attribute properties. All other properties need to be classified manually according to one of these 4 basic types or a lexicalisation and a new discourse schema need to be provided.

Once the information from the ontology is structured using the schemas, aggregation is performed to join similar RDF triples. This process joins adjacent triples that have the same first argument and have the same property name or if they are sub-properties of `attribute` or `part-whole` properties. For example, in the summary in Figure 3 we have 4 triples with the same first argument (the researcher) and all properties are attribute properties. Therefore, they are joined together as one proposition.

Without this aggregation step, there will be four separate sentences, resulting in a less coherent text:

> Kalina Bontcheva has a Dr appellation. Kalina Bontcheva has email K.Bontcheva@dcs.shef.ac.uk. Kalina Bontcheva has web page http://www.dcs.shef.ac.uk/ kalina/. Kalina Bontcheva has telephone number +4401142221930.

3.3 Extended Example

This section provides a step-by-step example first of how to customise ONTOSUM for an ontology, and next – of the generation process itself. The two processes can be repeated iteratively, i.e., the user can carry out some customisation, run ONTOSUM, analyse the problems, then go back to editing the lexicon or the ontology, etc.

The first stage in connecting an ontology to ONTOSUM is to execute the `Ontology2KBLex` component (see Figure 2) which generates the domain lexicon, as discussed in Section 3.1. For example, this would create the following entry, derived from the `full-name` property, which is one of the two property names (see Figure 2) used in the automatic creation of the lexical entries:

```
lex_entry_eng('K.Bontcheva.dcs.shef.ac.uk', 'Kalina Bontcheva',
                    [fs(pos,noun)]).
```

Once the lexicon has been completed, the next (optional) step is to introduce in the ontology property hierarchy the four linguistically-motivated properties discussed in Section 3.2. If the `has` heuristic has been enabled in

[6] http://www.aktors.org/ontology/
[7] http://ontoware.org/projects/swrc/

`Ontology2KBLex`, then some of the properties are classified automatically as attributive on the basis of their names.

In addition, if the `propertiesToFilterOut` parameter has been set, an ONTOSUM configuration file with this information is created automatically. This parameter enables the user to specify properties which should be filtered out from the textual summaries (see below for more detail).

At this stage, ONTOSUM is ready to be run on a given RDF/OWL description of an instance to produce its natural language summary. It is the responsibility of the application which calls ONTOSUM to choose which instance is to be described, i.e., to provide the RDF/OWL input. In the simplest case, this could be the user browsing the ontology, clicking on an instance, and asking for its textual summary.

For the sake of simplicity, throughout this example we will assume that there are no length restrictions for the summary and it will be generated as plain text. These issues are addressed in Section 4 next.

In this example we will use the AKT ontology and the RDF description of the author, part of which appears below:

```
<rdf:Description rdf:about="http://...#K.Bontcheva.dcs.shef.ac.uk">
  <ns0:family-name>Bontcheva</ns0:family-name>
  <ns0:full-name>Kalina Bontcheva</ns0:full-name>
  <ns0:given-name>Kalina</ns0:given-name>
  <ns0:has-appellation>Dr</ns0:has-appellation>
  <ns0:has-email-address>K.Bontcheva@dcs.shef.ac.uk</ns0:has-...>
  ...
  <rdf:type rdf:resource="http://...#Researcher-In-Academia"/>
</rdf:Description>
```

As shown in Figure 1, the first phase is *pre-processing*. Let us assume that there were no previous explanations, so no properties of this instance need to be removed to avoid repetition. During pre-processing ONTOSUM also removes properties given as values of the `propertiesToFilterOut` parameter of the Ontology2KBLex module. The function of this parameter is enable the user to exclude some information from the summary. For example, properties encoding the provenance of this instance or providing lexical information (e.g., full-name) may be excluded in this way.

In our example (see Figure 2), `full-name`, `family-name`, and `given-name` are specified as properties to be filtered out. Consequently, at the end of the pre-processing phase the input is transformed into:

```
<rdf:Description rdf:about="http://...#K.Bontcheva.dcs.shef.ac.uk">
  <ns0:has-appellation>Dr</ns0:has-appellation>
  <ns0:has-email-address>K.Bontcheva@dcs.shef.ac.uk</ns0:has-...>
  ...
  <rdf:type rdf:resource="http://...#Researcher-In-Academia"/>
</rdf:Description>
```

The next phase is *summary structuring*. Here we will consider two alternatives: one where the `has` heuristic has been enabled and one where it was disabled.

When the heuristic is enabled, the system would know which properties of the given instance are attribute properties, because their names start with has. Therefore the user would not need to specify their lexicalisations and the existing ONTOSUM schemas can be applied to order the triples. As we took a simple example containing only attribute properties, their order will not change, i.e., will remain as it was in the original input. However, as they are the same type of property, they will be aggregated into one semantic relation (ATTR) with several values – one for each property value. If there were other property types, then more semantic relations will be created and ordered according to the discourse schemas described in Section 3.2.

```
ATTR(Researcher-In-Academia: K.Bontcheva.dcs.shef.ac.uk,
     [   Appellation: Dr,
         string: K.Bontcheva@dcs.shef.ac.uk,
         string: +4401142221930,
         string: http://www.dcs.shef.ac.uk/~kalina/
     ]
```

The information that Dr is a value of type Appellation and email, telephone, and URL are strings comes from the range restrictions in the property definitions in the ontology, e.g., see the has-email-address definition in Section 3.1. Also, the lexical entry for `K.Bontcheva.dcs.shef.ac.uk` is used instead of the unique identifier from the ontology.

Given this input, the HYLITE+ generator will verbalise it as:

> Kalina Bontcheva has a Dr appellation, K.Bontcheva@dcs.shef.., http://www.dcs.shef.ac.uk/ kalina/, and +4401142221930.

The problem with this summary comes from the fact that the ontology engineer decided to encode some of the information about researchers using classes (e.g., Appellation), while the rest is encoded as datatype properties with range `string`. Since this is not an ontology class, the generator only provides its value (e.g., +4401142221930), but it lacks the information that this is a telephone number, as this is only encoded implicitly in the property name – `has-telephone-number`.

One solution is to modify the ontology by introducing the required classes (Email, TelephoneNumber, etc.) and changing the property ranges from string to these new classes. This would have the benefit of making explicit the semantics of these properties and their values. However, it may not always be desirable to modify the ontology.

The second solution is to provide the generator with manually written mapping rules which map the ranges of given properties to their lexical classes, e.g., `lex-mapping(has-email, string, email)`. Then, using these mappings, ONTOSUM will produce instead:

```
ATTR(Researcher-In-Academia: K.Bontcheva.dcs.shef.ac.uk,
     [   Appellation: Dr,
         email: K.Bontcheva@dcs.shef.ac.uk,
         telephone number: +4401142221930,
         web page: http://www.dcs.shef.ac.uk/~kalina/
     ]
```

The disadvantage of the second approach is that it requires the user to create manually these mappings for each problematic datatype property. However, the number of such properties is often quite small and, in our experience, it is feasible to do that in cases when the ontology itself cannot be modified.

The third approach is to not define these properties as `attribute` properties, i.e., to disable the `has` heuristic. In that case, ONTOSUM would use instead the lexicalisations of the properties themselves, derived automatically from their definitions (see Section 3.1). The disadvantage of this approach is that the structuring module will not be able to aggregate the four statements, as they will involve four different properties:

```
has-appellation(Researcher-In-Academia: K.Bontcheva.dcs.shef.ac.uk,
                    Appellation: Dr)
has-email-address(Researcher-In-Academia: K.Bontcheva.dcs.shef.ac.uk,
                    string: K.Bontcheva@dcs.shef.ac.uk)
has-telephone-number(Researcher-In-Academia: K.Bontcheva.dcs.shef.ac.uk,
                    string: +4401142221930)
has-web-address(Researcher-In-Academia: K.Bontcheva.dcs.shef.ac.uk,
                    string: http://www.dcs.shef.ac.uk/~kalina/)
```

Consequently, they will be verbalised as four separate sentences:

> Kalina Bontcheva has a Dr appellation. Kalina Bontcheva has email K.Bontcheva@dcs.shef.ac.uk. Kalina Bontcheva has web page http://www.dcs.shef.ac.uk/ kalina/. Kalina Bontcheva has telephone number +4401142221930.

In this case the information that K.Bontcheva@dcs.shef.ac.uk is an email address comes from the lexicalisation of the property `has-email-address` (see Section 3.1). The same is true for the other datatype properties.

However, while the problem with the implicit semantics of the datatype properties has been solved, the resulting summary is no longer so concise. One solution, to be implemented in future work, would be to implement a syntactic aggregation component which merges two sentences when they have the same subject and verb.

4 Summary Tailoring

The types of tailoring/personalisation considered here are based on information from the user's device profile. Most specifically, we looked into generating summaries within a given length restriction (e.g., 160 characters for mobile phones)

and different formats – HTML for browsers and plain texts for emails and mobile phones.

4.1 Choosing Formatting

Hypertext usability studies [6] have shown that formatting is very important since it improves readability. Bullet lists and font size in particular facilitate skimming by making important information more prominent. Therefore our work focused on generating lists, while font size was made customisable by the user by using the browser's chosen size. The use of HTML lists in our system is determined as a first step of the text generation process, on the basis of the fully fledged text plan.

The semantic aggregation stage joins all propositions which share the same focused entity and relation, so the resulting more complex propositions can have three or more entities that need to be enumerated in the same sentence. For example, the following complex proposition appears when generating summaries of researchers' contact details and activities (shown below as a conceptual graph):

```
[RESEARCHER :fs(focus, true)] <- (HAS) <- [EMAIL: xxx]
                              - (HAS) <- [WEB-PAGE: yyy]
                              - (HAS) <- [TELEPHONE: zzz].
```

The formatter module in HYLITE+ examines each proposition in the text plan to determine if the focused entity participates in the same relation with three or more different concepts. If no formatting is required, then a conjunction will be generated (see example in the previous section).

In the case of HTML summary, such propositions are annotated for bullet list formatting if the repeating relations are `ISA, PART_OF`, or `HAS`). No HTML markup is generated at this stage. Instead, the formatting choice is stored as metadata on the proposition:

```
[RESEARCHER :fs(focus, true)] <- (HAS) <- [EMAIL: xxx]
                              - (HAS) <- [WEB-PAGE: yyy]
                              - (HAS) <- [TELEPHONE: zzz]. : fs(format,ul)
```

This information is used later by the grammar to generate the HTML tags at the same time as the text itself (see Figure 3). This separation allows the formatter (or a later module) to change this choice if it is not appropriate, e.g., to avoid overusing lists.

4.2 Controlling Summary Length

In general, there are two ways in which size constraints can be taken into account during summary generation: (i) approximate the length during content planning; or (ii) given a text plan, decide how to modify or verbalise it to the given size limit via revision. As shown by [9], approximating the text length during content planning is possible but suffers from two problems. The first one is that the result is not exact, so when formatting is added later the text might not fit into the

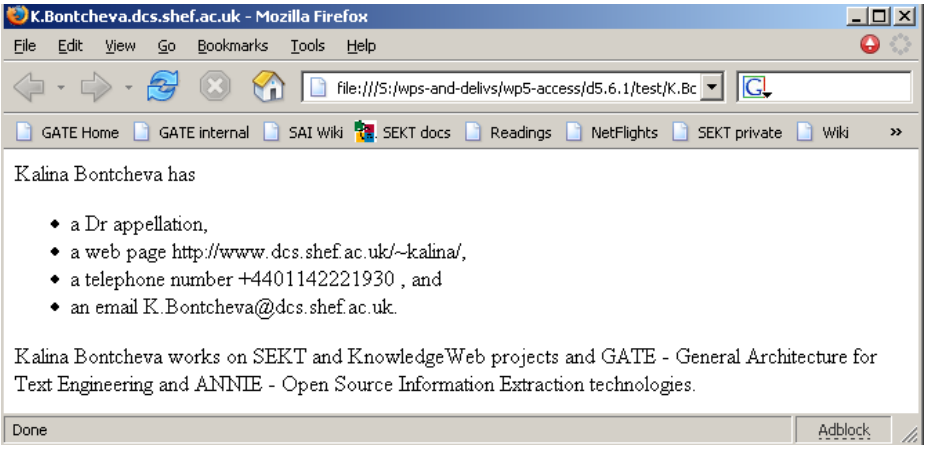

Fig. 3. The HTML summary generated from the RDF input from Section 3.3 with no length restrictions

page any more. The second, more significant problem with this method is that it is hard to maintain and update by non-experts.

Another alternative is to implement a text revision mechanism to analyse the content and structure of the generated summary. However, despite the gains in fluency and coherence, revision-based approaches tend to suffer from computational problems due to the large number of alternatives that need to be explored.

The requirements towards our system are computational efficiency and easy modification by NLG experts. In addition, while the size limit is important, it is not critical if it were exceeded slightly. Therefore, our system always puts in the text structure all statements from the RDF/OWL input. Then given such a text plan, the surface realiser generates the sentences one by one and when a new sentence is added to the summary, summary length is incremented accordingly. As soon as the value of this variable plus the length of the next sentence exceed the limit, the surface realiser is not called further and the last sentence is not included.

The desired summary length is supplied as an input parameter to ONTOSUM. For instance, when the length restriction is set to 160 characters (and plain text), the generated summary is:

> Kalina Bontcheva has a Dr appellation, a web page http://www.dcs.shef.ac.uk/k̃alina/, a phone number +4401142221930 , and an email address K.Bontcheva@dcs.shef.ac.uk.

In contrast, when there are no length restrictions and the target formatting is HTML, a longer summary is generated (see Figure 3).

5 Using Ontology Mapping to Run ONTOSUM on Different Ontologies

In the previous sections we discussed how ONTOSUM is adapted to a new ontology. However, frequently there is more than one ontology describing the same or similar domains [4]. For example, both the AKT and SWRC ontologies, discussed above, have concepts describing researchers, their publications, contact details, etc. Therefore, having customised ONTOSUM to the AKT ontology, instead of adapting it to SWRC from scratch, one could use *ontology mapping* rules [4] to "translate" the SWRC instance desctiptions into AKT ones and then run ONTOSUM without modifications.

In order to experiment with this approach, we designed manually a set of mapping rules for concepts and properties in the two ontologies. Some concept mappings are: `swrc:AssistantProfessor` is mapped to `akt:Lecturer-In-Academia`, `swrc:AssociateProfessor` – to `akt:Senior-Lecturer-In-Academia`, etc.

Respectively, some property mappings are: `swrc:name` – `akt: full-name`, `swrc:phone` – `akt:has-telephone -number`, `swrc:fax` – `akt:has-fax-number`, `swrc:homepage` – `akt:has-web-address`. Some SWRC properties, e.g., `photo` do not have a corresponding property in the AKT ontology. Therefore, no mapping was provided for them and, consequently, they are not included in the generated summaries.

Once defined, the mapping rules are applied to transform automatically instance descriptions from the SWRC to the AKT ontology, prior to sending them to ONTOSUM for generation. For example, the SWRC instance describing York Sure[8] looks as follows:

```
<rdf:Description rdf:about="http://www.aifb.uni-
          karlsruhe.de/Personen/viewPersonOWL#instance?id_db=20">
 <rdf:type>
  <owl:Class rdf:about="&swrc;AssistantProfessor"/>
 </rdf:type>
 <swrc:name rdf:datatype="&xsd;string">York Sure</swrc:name>
 <swrc:phone rdf:datatype="&xsd;string"> +49 (0) 721 608 6592
 </swrc:phone>
 <swrc:fax rdf:datatype="&xsd;string"> +49 (0) 721 608 6580
 </swrc:fax>
 <swrc:homepage rdf:datatype="&xsd;string">
   http://www.aifb.uni-karlsruhe.de/WBS/ysu
 </swrc:homepage>
</rdf:Description>
```

After applying the mapping rules and removing properties for which no mappings exist, the system obtains:

```
<rdf:Description rdf:about="http://www.aifb.uni-
```

[8] The author is grateful to York Sure for supplying the SWRC instance data.

```
            karlsruhe.de/Personen/viewPersonOWL#instance?id_db=20">
  <rdf:type>
   <owl:Class rdf:about="http...#Lecturer-In-Academia"/>
  </rdf:type>
  <akt:full-name rdf:datatype="&xsd;string">York Sure</akt:full-name>
  <akt:has-telephone-number rdf:datatype="..."> +49 (0) 721 608 6592
  </akt:has-telephone-number>
  <akt:has-fax-number rdf:datatype="..."> +49 (0) 721 608 6580
  </akt:has-fax-number>
  <akt:has-web-address rdf:datatype="&xsd;string">
    http://www.aifb.uni-karlsruhe.de/WBS/ysu
  </akt:has-web-address>
 </rdf:Description>
```

When this input is passed to ONTOSUM, it generates the following textual summary, without requiring any customisation:

> York Sure has a telephone number +49 (0) 721 608 6592, a fax number +49 (0) 721 608 6580 , and a web page http://www.aifb.uni-karlsruhe.de/WBS/ysu.

The advantages of using ontology mapping to enable ONTOSUM to run on different ontologies are: *(i)* no ONTOSUM customisation is required by the user; *(ii)* ontology mapping can be performed by ontology engineers and there are even some tools that automate parts of this process [4].

A future extension of this approach would be to allow for more sophisticated mapping or even *ontology merging*, in order to enable ONTOSUM to verbalise also properties and concepts which do not exist in the original ontology. In this case, some limited customisation will be required, mainly concerned with providing new lexical information.

6 Conclusion

This paper presented the ONTOSUM system which uses Natural Language Generation (NLG) techniques to produce textual summaries from Semantic Web ontologies. The main contribution of this work is in showing how existing NLG tools can be adapted to take Semantic Web ontologies as their input, in a way which minimises the customisation effort while being more flexible than template-based ontology verbalisers (e.g., [11]).

A major factor in the quality of the generated summaries is the content of the ontology itself. For instance, the use of string datatype properties with implicit semantics (e.g., `has-web-address`) leads to the generation of summaries with missing semantic information. Three approaches to overcome this problem were presented here and users can choose the one that suits their application best.

We also showed how the generated summaries can be tailored for formatting and length restrictions from a device profile (e.g., mobile phone, Web browser). Another innovative idea is the use of ontology mapping to enable ONTOSUM to generate text from different ontologies, without customisation effort.

Future work will focus on the creation of a user-friendly tool for specifying new summary structuring schemas, because at present this is done directly in the generator's internal structures, which are hard to understand for non-specialists. Another strand of this work will aim at further investigation of the use of ontology mapping and merging in NLG systems.

Another major area for future work is system evaluation. NLG systems are normally evaluated with respect to their usefulness for a particular (set of) task(s), which is established by measuring user performance on these tasks, i.e., *extrinsic* evaluation. This is often also referred to as *black-box* evaluation, because it does not focus on any specific module, but evaluates the system's performance as a whole. Therefore we plan to carry out empirical studies with end-users as part of the SEKT digital library case study. The goal is to carry out a qualitative evaluation of the textual summaries when they appear within a complete semantically-enabled knowledge management system.

References

1. G. Aguado, A. Bañón, John A. Bateman, S. Bernardos, M. Fernández, A. Gómez-Pérez, E. Nieto, A. Olalla, R. Plaza, and A. Sánchez. ONTOGENERATION: Reusing domain and linguistic ontologies for Spanish text generation. In *Workshop on Applications of Ontologies and Problem Solving Methods, ECAI'98*, 1998.
2. K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham. Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, 10(3/4):349—373, 2004.
3. 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)*, 2004.
4. J. de Bruijn, F. Martin-Recuerda, D. Manov, and M. Ehrig. State-of-the-art survey on Ontology Merging and Aligning v1. Technical report, SEKT project deliverable D4.2.1, 2004. http://sw.deri.org/ jos/sekt-d4.2.1-mediation-survey-final.pdf.
5. Kathleen R McKeown. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press, 1985.
6. Jakob Nielsen. *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, 2000.
7. Cécile L. Paris. Tailoring object descriptions to the user's level of expertise. *Computational Linguistics*, 14 (3):64–78, September 1988. Special Issue on User Modelling.
8. E. Reiter and R. Dale. *Building Natural Language Generation Systems*. Cambridge University Press, Cambridge, 2000.
9. Ehud Reiter. Pipelines and size constraints. *Computational Linguistics*, 26:251–259, 2000.
10. J.F. Sowa, editor. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, California, San Mateo, CA, 1991.
11. G. Wilcock. Talking OWLs: Towards an Ontology Verbalizer. In *Human Language Technology for the Semantic Web and Web Services, ISWC'03*, pages 109–112, Sanibel Island, Florida, 2003.
12. 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*, pages 58–63, Acapulco, 2003.