ITALIAN NATIONAL RESEARCH COUNCIL
"NELLO CARRARA" INSTITUTE FOR APPLIED PHYSICS
CNR FLORENCE RESEARCH AREA
Italy

TECHNICAL, SCIENTIFIC AND RESEARCH REPORTS

Vol. 2 - n. 65-9 (2010)

Francesco Gabbanini

Exploiting web resources for the identification of relations between concepts: a Java based implementation and case study

CNR-IFAC-TR-10/010

ISSN 2035-5831

Consiglio Nazionale delle Ricerche
Istituto di Fisica Applicata "Nello Carrara"

VIA MADONNA DEL PIANO 10 - 50019 SESTO FIORENTINO – ITALIA

CNR-IFAC-TR-10/010

Francesco Gabbanini

Exploiting web resources for the identification of relations between concepts: a Java based implementation and case study

Commessa ICT-P10-007
e-Inclusion Lab

Exploiting web resources for the identification of relations between concepts: a Java based implementation and case study

F. Gabbanini

Institute for Applied Physics, Italian National Research Council
Via Madonna del Piano 10, 50019, Sesto Fiorentino, Firenze, Italy
f.gabbanini@ifac.cnr.it

1 Introduction

A Java based implementation of an Ontology Evolution Manager was described in Gabbanini (2010): it is a framework offering a set of tools to support processes of manipulation and growth of ontological knowledge bases, based on inputs consisting in free text documents.

The Ontology Evolution Manager can be used to support the process of Ontology Evolution, i.e., the process of identifying potential novel entities and relationships to be included in an established ontology.

This report describes a Java based application, built using the Ontology Evolution Manager, intended to perform ontology evolution processes, by enriching ontologies with new relations. The enrichment phase uses as sources of background knowledge the WordNet repository (see WordNet, 2010) and the Scarlet system (Sabou et al., 2008, Sabou et al., 2008b, Scarlet, 2010). The application is based on ideas described in Zablith et al. (2009), but new ideas have been introduced and the code has been implemented from scratch by the author, so as to be reusable within the framework of the *Collective Knowledge Management System* described by Burzagli et al. (2010).

The report describes techniques and implementation details, along with a test case in which an ontology, built within the e-Inclusion Laboratory[1] to describe the domain of inclusive tourism, is enriched with entities and relationships generated from the analysis of textual reviews, contributed by customers of a real web based service that allows booking and commenting on the accessibility of a selection of accommodation resources all over the world.

2 The entity and relation extraction engine

The entity and relation extraction engine described in this report is a system that takes advantage of background information to identify new entities and relations, which are then used to enrich ontologies. Such an engine was implemented using the Ontology Evolution Manager framework described in Gabbanini (2010).

In order to achieve its aims, the engine goes through three fundamental steps, which are represented by:

1. Identifying key terms in a given corpus of text documents;
2. Check for relationships between the identified terms and entities that are present in a reference ontology;
3. Add novel entities and relationships to the ontology.

[1] See http://eilab.ifac.cnr.it, last visited on 27/10/2010.

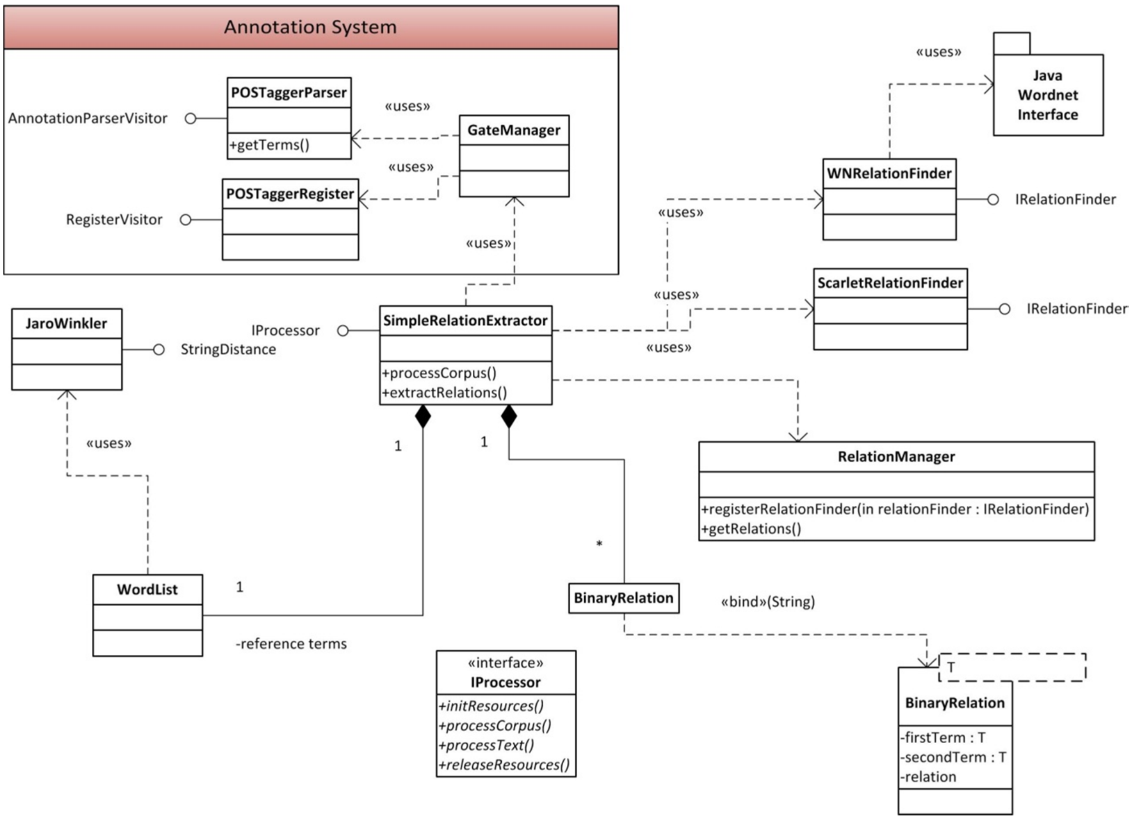

Figure 1. UML class diagram of the Ontology Evolution Manager.

This 3-step process has been modelled using classes that are structured according to the UML diagram in Figure 1. When reading this paper, it is to be noted that references are sometimes made to classes and interfaces which in Figure 1 are contained within the box named *Annotation System* (top left corner of Figure 1): for a more detailed description of them, the interested reader should refer to Gabbanini (2010).

The central point of the relation identification and extraction process is represented by the `SimpleRelationExtractor` class, which is able to process a corpus of text documents, extract terms and relate them to terms contained in a given reference terms set, by using relation manager objects, described later on in the report.

The reference terms set consists in terms representing concepts that are contained in the ontology which is to be enriched through the evolution process. This means that it is necessary to start up the process with a valid ontology which, in the case examined by this report, was set up by the e-Inclusion Lab by taking advantage of work done by a team of professionals in the field of accessible tourism, during the EU CARE project[2]. The ontology is meant to describe physical characteristics of inner spaces of touristic accommodations: while its building process is out of the scope of this report, an excerpt of the resulting ontology, which was used for testing purposes, is shown in Figure 2.

[2] See http://www.interreg-care.org/site/, the site was last visited on 12/10/2010. As of 26/10/2010 it seems to be down for maintenance.

3 The evolution process

The entry point for the evolution process is represented by the `SimpleRelationExtractor` class (see code in Table 2), which allows using an arbitrary set of relation finding engines through the use of a visitor pattern (Gamma et al., 1995), as described in section 3.1. Specifically, the starting point for the relation discovery process is the `extractRelations` method, taking as an input a set of reference terms, which is a set of `String` objects, designed to detect similarities between strings in such a way that, for example, "river" and "rivrer" are treated as being the same word: this strategy is adopted to check that no identical or highly similar terms are contained in the list and also allows accounting for spelling or typing errors. In order to obtain this behaviour, the set of reference terms is implemented using an object of `WordSet` class (see code in Table 3). `WordSet` extends the `HashSet` class, of which it overrides the `contains` method by making use of a suitable string distance function.

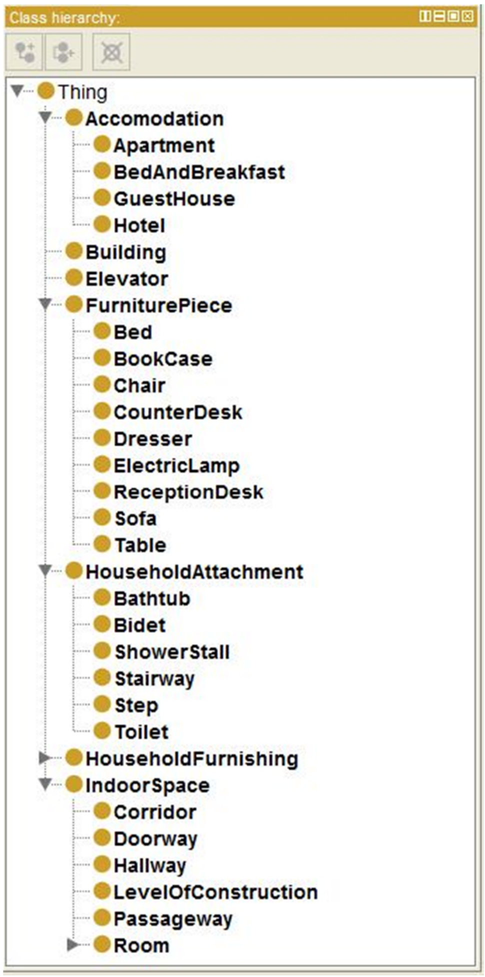

Figure 2. Excerpt from the test ontology, describing the tourism domain.

The string distance is computed using a class implementing the `StringDistance` interface, as defined in the *SecondString* library (SecondString, 2010). This library is an open-source Java-based package of approximate string-matching techniques, developed by researchers at Carnegie Mellon University from the Center for Automated Learning and Discovery, the Department of Statistics, and the Center for Computer and Communications Security. The package contains a wide range of implementations of string distance functions: the chosen distance function for the integration within the `SimpleRelationExtractor` was the Jaro-Winkler metric, described in Winkler (1999), which is in turn a refinement of the distance described in Jaro (1995). The distance is available from the SecondString library's `JaroWinkler` class.

3.1 Parsing the corpus for new terms

In order to search for relations, the `SimpleRelationExtractor` needs to be provided with a set of terms to match with those in the reference list. This is achieved by taking advantage of the `GateManager` class. The rest of this section describes the process more in depth.

Firstly, the `SimpleRelationExtractor` class is designed to implement an `IProcessor` interface. This interface has been introduced to identify any resource that uses text processing functionalities provided by the GATE framework. Its implementation requires specifying four fundamental methods:

- `initResources`: allows to initialize the Natural Language Processing system, by setting up appropriate plugins and resources;
- `processCorpus`: allows to process a corpus of textual documents;
- `processText`: allows to process a string of text;
- `releaseResources`: allows to release language processing resources.

As for the initialization of natural language processing resources, the process is supported by the *Annotation System* through the use of registers implementing the `RegisterVisitor` interface. The processing phase produces annotations that are parsed through parser classes implementing the `AnnotationParserVisitor` interface (for more details see Gabbanini (2010)). In order to obtain suitable annotations for the relation discovery process, a new register and a new parser were written and added to the *Annotation System*. The `POSTaggerRegister` serves to annotate text with information regarding which part of speech the various words contained in the text documents represent, while the `POSTaggerParser` is meant to interpret those annotations and to filter them in order to obtain sets of words representing relevant parts of speech (for example, for obtaining only the adverbs, or only the verbs and proper nouns etc…).

This mechanism was used to extract singular and plural nouns from the text corpus that was chosen to provide input to the ontology evolution process. These are the terms for which relations with existing concepts in the ontology have to be investigated.

3.2 Finding relations: the `RelationManager`

The process of finding relations is handled by the `RelationManager` class (see code in Table 4), and is based on a visitor pattern. The `RelationManager` allows registering instances of different classes that are capable of performing relation discovery between terms. Each class has to implement the `IRelationFinder` interface. For the purpose of the example described in this report, two classes were designed to be employed for the relation discovery process: the `WNRelationFinder` class (see code in Table 5) and the `ScarletRelationFinder` class (see Figure 1, top right corner). The latter is based on Scarlet (see Sabou et al., 2008), a third party Java software library implementing techniques for discovering relations between two concepts by harvesting the Semantic Web, i.e., by automatically finding and exploring multiple and heterogeneous online ontologies.

The `WNRelationFinder` class was written from scratch and is built to exploit WordNet *synsets* and the *hypernym*/*hyponym-holonym*/*meronym* concepts. Actually, WordNet (see WordNet, 2010) is a lexical database that, for each English word, is able to provide a set of synonyms called *synsets*. Each synset is related to other synsets by a number of semantic relations. These relations vary based on the type of word, and include, in the case of nouns:

- *hypernym*: Y is a hypernym of X if every X is a (kind of) Y (canine is a hypernym of dog, because every dog is a member of the larger category of canines)
- *hyponym*: Y is a hyponym of X if every Y is a (kind of) X (dog is a hyponym of canine)
- *instance hypernym*: Y is a instance-hypernym of X if X is an instance of Y (author is an instance hypernym of Jane Austen);
- *instance hyponym*: Y is a instance-hyponym of X if Y is an instance of X (Jane Austen is an instance hyponym of author);
- *holonym*:

- *part*: Y is a part-holonym of X if X is a part of Y (building is a holonym of window)
    - *member*: Y is a member-holonym of X if X is a member of Y (faculty is a member holonym of professor);
    - *substance*: Y is a substance-holonym of X if X is a substance of Y (bread is a substance holonym of flour)
- *meronym*:
    - *part*: Y is a part-meronym of X if Y is a part of X (window is a meronym of building)
    - *member*: Y is a member-meronym of X if Y is a member of X (professor is a member meronym of faculty);
    - *substance*: Y is a substance-meronym of X if Y is a substance of X (flour is a substance meronym of bread).

Semantic relations between WordNet synsets are used to derive new relations between terms in the ontology and terms in the text corpus.

In order to access WordNet and navigate its database, the Java Wordnet Interface (JWI, see JWI, 2010), developed by the MIT, was used. The `WNRelationFinder` uses the JWI within a recursive algorithm that was designed to walk across the semantic relations tree until it finds that one of the previously listed relations is involving two given terms. When the process ends up, the relation finder returns a list of `BinaryRelation` objects, each relating a pair of terms (one from the corpus and one representing an entity in the ontology) according to a specified relation. It is then the responsibility of the Ontology Persistence Layer to translate relations into valid RDF and OWL statements that enrich the ontology.

An excerpt of the code that implements the relation discovery process is given in Tables 1 to 5.

Table 1. Excerpt of the code implementing the overall relation extraction process.

```
@Test
public class OntoEvolutionTest {

private HashSet<String> owlEntities;
private final String baseURI = "...";

@Before
public void setUp() throws Exception {
  getOWLEntities();
}

public void testAddRelations() throws (...) {
  List<BinaryRelation<String>> relations = extractRelations();
  OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
  OWLOntology owlOntology = manager.loadOntologyFromOntologyDocument(...);
  OWLDataFactory owlDataFactory = manager.getOWLDataFactory();
  AxiomManager axiomManager = new AxiomManager(owlDataFactory, baseURI);
  for(BinaryRelation<String> relation : relations) {
    String t1 = relation.getFirstTerm();
    String t2 = relation.getSecondTerm();

    Set<AddAxiom> axioms = axiomManager.getAxiom(owlOntology, t1, t2,
relation.getRelation());

    for(AddAxiom axiom : axioms) {
      manager.applyChange(axiom);
    }

    OutputStream outputStream = new FileOutputStream("...");
```

```
      manager.saveOntology(owlOntology, outputStream);
    }
  }

  private List<BinaryRelation<String>> extractRelations() throws (...) {
    SimpleRelationExtractor relationExtractor = new SimpleRelationExtractor();
    relationExtractor.processCorpus(corpusPath, corpusExt);
    List<BinaryRelation<String>> relations =
relationExtractor.extractRelations(owlEntities);
      return relations;
  }

  private void getOWLEntities() throws OWLOntologyCreationException {
    //loads entities from the ontology (code not shown)
  }

  }
```

Table 2. Implementation details: the `SimpleRelationExtractor` and `WNRelationSet` classes. It is to be noted that *meronym* and *hyponym* relations are exploited by symmetry.

```
public class SimpleRelationExtractor implements IProcessor {
private WordSet referenceTermList;
private List<BinaryRelation<String>> relations;

@Override
public void initResources() throws (...) {
  GateManager.getInstance().registerPlugin("ANNIE");
  GateManager.getInstance().registerResource(new AnnotationDeleteRegister());
  GateManager.getInstance().registerResource(new SentenceSplitterRegister());
  GateManager.getInstance().registerResource(new DefaultTokenizerRegister());
  GateManager.getInstance().registerResource(new POSTaggerRegister());
}

@Override
public void processCorpus(String pathName, String ext) throws (...) {
  initResources();
  ...
  GateManager.getInstance().elaborateCorpus();
}

public List<BinaryRelation<String>> extractRelations(Collection<String>
referenceTerms) throws (...) {
  referenceTermList = new WordSet(new JaroWinkler());
  referenceTermList.addAll(referenceTerms);

  Set<String> extractNouns = extractNouns();
  relations = new ArrayList<BinaryRelation<String>>();

  for(String noun : extractNouns) {
    if(noun.length() > 2) {
      findRelations(noun);
    }
  }
  return relations;
}
private void findRelations(String noun) throws (...) {
  if(referenceTermList.contains(noun)) {
    return;
  }
  for(String refString : referenceTermList) {
    findRelationAbout(refString, noun);
  }
}
private void findRelationAbout(String referenceTerm, String noun) throws (...) {
```

```
    List<IWord> iWords = JWI.getInstance().getIWords(referenceTerm);
    if(iWords == null) return;
    IWord ontoLabelWord = iWords.get(0);

    iWords = JWI.getInstance().getIWords(noun);
    if(iWords == null) return;
    IWord nounWord = iWords.get(0);

    RelationManager relationManager = new RelationManager();
    WNRelationFinder wnRelationFinder = new WNRelationFinder(ontoLabelWord, nounWord,
  new WNRelationSet().initDefaults());

    relationManager.registerRelationFinder(wnRelationFinder);
    relationManager.registerRelationFinder(new ScarletRelationFinder(ontoLabel, noun));

    if (relationManager.getRelations() != null && relationManager.getRelations().size()
  > 0) {
      relations.addAll(relationManager.getRelations());
    }

  }
  }

  public class WNRelationSet implements Iterable<IPointer> {
  private HashSet<IPointer> relationPointers = new HashSet<IPointer>();

  public void addPointer(IPointer p) {
    relationPointers.add(p);
  }
  public WNRelationSet initDefaults() {
    relationPointers.clear();
    relationPointers.add(Pointer.HOLONYM_MEMBER);
    relationPointers.add(Pointer.HOLONYM_PART);
    relationPointers.add(Pointer.HOLONYM_SUBSTANCE);

    relationPointers.add(Pointer.HYPERNYM);
    relationPointers.add(Pointer.HYPERNYM_INSTANCE);

    return this;
  }
  @Override
  public Iterator<IPointer> iterator() {
    return relationPointers.iterator();
  }
  }
```

Table 3. Implementation details: the `WordSet` class

```
  public class WordSet extends HashSet<String> {

  private StringDistance distance;
  private float threshold;
  public WordSet(StringDistance distance) {
    this.distance = distance;
    threshold = 0.95f;
  }

  @Override
  public boolean contains(Object noun) {
    if(distance == null)
      return super.contains(noun);
    for(String s : this) {
      if (distance.score(s, noun.toString()) > threshold) {
        return true;
      }
    }
```

```
    return false;
  }

  public String find(String noun) {
    for(String s : this) {
      if (distance.score(s, noun.toString()) > threshold) {
        return s;
      }
    }
    return null;
  }
  }
```

Table 4. Implementation details: the `RelationManager` class

```
  public class RelationManager {

  List<BinaryRelation<String>> relations = new ArrayList<BinaryRelation<String>>();
  public void registerRelationFinder(IRelationFinder relationFinder) throws
  RelationFinderException {
       relationFinder.visit(this);
  }
  public void addRelation(BinaryRelation<String> relation) {
    relations.add(relation);
  }
  public List<BinaryRelation<String>> getRelations() {
    return relations;
  }
  }
```

Table 5. Implementation details: the `WNRelationFinder` class

```
  public class WNRelationFinder implements IRelationFinder {
  private IWord originalWord;
  private IWord secondWord;
  private WNRelationSet relationSet;

  public WNRelationFinder(IWord originalWord, IWord secondWord, WNRelationSet
  relationSet) {
    this.originalWord = originalWord;
    this.secondWord = secondWord;
    this.relationSet = relationSet;
  }

  @Override
  public void visit(RelationManager relationFinder) throws RelationFinderException {
    for(IPointer relPointer : relationSet) {
      exploitRelation(relationFinder, relPointer);
    }
  }

  private void exploitRelation(RelationManager relationFinder, IPointer relPointer)
    throws RelationFinderException {

    try {
      List<BinaryRelation<IWord>> relations = findRelations(originalWord, secondWord,
  relPointer);
      for(BinaryRelation<IWord> binaryRelation : relations) {
        BinaryRelation<String> stringRelation = new
  BinaryRelation<String>(binaryRelation.getFirstTerm().getLemma(),
  binaryRelation.getSecondTerm().getLemma(), binaryRelation.getRelation());
        relationFinder.addRelation(stringRelation);
      }
```

```
    } catch (MalformedURLException e) {
      throw new RelationFinderException(e.getMessage());
    }
  }

  private List<BinaryRelation<IWord>> findRelations(IWord fWord, IWord sWord, IPointer
    relPointer) throws MalformedURLException, RelationFinderException {

    List<BinaryRelation<IWord>> relations = new ArrayList<BinaryRelation<IWord>>();

    SynsetHierarchyBuilder hierarchyBuilder = new SynsetHierarchyBuilder();
    SynsetHierarchy synsetHierarchy1 = hierarchyBuilder.build(fWord, relPointer);

    //is "sWord" in the synset of type "pointer" of "word"?
    BinaryRelation<IWord> relation = findRelations(fWord, sWord, synsetHierarchy1);
    if (relation != null) {
      relations.add(relation);
      return relations;
    }

    SynsetHierarchy synsetHierarchy2 = hierarchyBuilder.build(sWord, relPointer);

    //is "word" in the synset of type "pointer" of "sWord"?
    relation = findRelations(sWord, fWord, synsetHierarchy2);
    if (relation != null) {
      relations.add(relation);
      return relations;
    }

    return relations;

  }

  private BinaryRelation<IWord> findRelations(IWord firstWord, IWord secondWord,
    SynsetHierarchy synsetHierarchy) throws RelationFinderException {

    ISynset sSynset = secondWord.getSynset();
    for(ISynset synset : synsetHierarchy) {
      if(synset.equals(sSynset)) {
        return buildRelation(firstWord, secondWord, synsetHierarchy.getPointerType());
      }
    }
    return null;
  }

  private BinaryRelation<IWord> buildRelation(IWord firstRelationTerm, IWord
    secondRelationTerm, IPointer pointerType) throws RelationFinderException {

    //builds an appropriate BinaryRelation...
  }
  }
```

4 A sample test case

In order to evaluate the correctness and validity (at least, from a technical point of view) of the approach, a sample application was setup in which the ontology introduced in section 3 (see Figure 2 for an excerpt) is to be enriched by inspection of a corpus of text documents consisting in 88 user generated reviews, taken from the website `http://www.accessatlast.com`. Each review is expressed as free text and reflects the opinion of a user regarding an accommodation that s/he has stayed in. It is to be noted that the example does not use the relation finding engine based on Scarlet, but only the one based on WordNet.

A `POSTaggerParser` object was used to parse the 88 reviews, in order to provide the system with a list of 779 terms (after filtering out for similarities, see section 3), which represent candidate terms for relation discovery. These terms are then matched with terms denoting entities contained in the ontology. In this way, the relation discovery engine discovers 42 relations, of which 15 are *part meronym* relations and the rest are *hyponym* relations. It is to be noted that in this example, only the first WordNet synset of each term, which represent the most common (according to WordNet statistics) sense in which the term itself is used, is exploited for the relation discovery process.

Table 6. Relations identified by the `SimpleRelationExtractor` after parsing a corpus of 88 reviews from `http://www.accessatlast.com`.

Term	Relation	Term	Term	Relation	Term
Barn	subClassOf	Building	Sofa	subClassOf	Furniture
Architecture	subClassOf	Building	Dresser	subClassOf	Furniture
Cottage	subClassOf	Building	Bed	subClassOf	Furniture
Castle	subClassOf	Building	House	subClassOf	Building
Bar	subClassOf	Room	GuestHouse	subClassOf	House
Chalet	subClassOf	Building	Restaurant	subClassOf	Building
Chair	subClassOf	Furniture	Resort	subClassOf	Building
BookCase	subClassOf	Furniture	Resort	subClassOf	Hotel
Table	subClassOf	Group	Hospital	subClassOf	Building
Stairs	subClassOf	Stairway	Wheelchair	subClassOf	Chair
Wall	partOf	Building	Wall	partOf	Room
Doorway	partOf	Wall	Wall	partOf	Hallway
Wall	partOf	Hall	Garage	subClassOf	Building
Door	partOf	Building	Door	partOf	Room
Door	partOf	Doorway	Door	partOf	Hallway
Door	partOf	Hall	Carport	subClassOf	Building
Tub	partOf	Bathroom	Towel	subClassOf	Piece
Floor	partOf	Building	Floor	partOf	Room
Floor	partOf	Hallway	Floor	partOf	Hall
Doorway	subClassOf	Entrance	Step	subClassOf	Selection
solarium	subClassOf	Room	Sauna	subClassOf	Room

Table 6 lists all the newly discovered relations that link novel terms to existing entities in the ontology. Regarding the insertion of new relations into the ontology, when two terms X and Y are discovered to be related by a *subClassOf* relation, a corresponding *rdfs:subClassOf* assertion is built to enrich the ontology. When X and Y are related by a *partOf* relation, an object property *isPartOf*, whose domain is X and whose range is Y, is added to the ontology.

While most of the all the triples listed in Table 6 define statements that seem to be consistent with the given context and the given domain of interest (i.e., describing the physical characteristics of accommodations), the ones with a grey background in Table 6 merit attention:

- The term *Table* is related to *Group*, which does not seem to be a good fit for the domain, probably due to the fact that one of the WordNet synsets of Table has the meaning “a company of people assembled at a table for a meal or game”; in this case it would have been probably better not to add the triple at all;
- The term *Step* is related to *Selection*, because it is taken in the sense of “any maneuver made as part of progress toward a goal”; again, the relation does not fit the particular context under study.

It is also interesting to note that a set of entities (*Sofa*, *Dresser*, *Bed*, *Chair*, *BookCase*) are related to the term *Furniture* through the *subClassOf* relation, and they were already related to *FurniturePiece* in the ontology, through the same relation: in such cases (i.e., when X *subClassOf* Y and X *subClassOf* Z) it would be interesting to set up a procedure to check for some kind of relation between Y and Z. In this particular case it would probably be an equivalence relation as the two terms are synonyms.

5 A refinement of the relation extraction process

The results highlighted in Table 6 and discussed in section 4 were a starting point from which a new refined version of the relation extractor was implemented. The main driving idea for the implementation was to avoid getting relations which are plainly and noticeably “out of context”, which means that the relation finding engine takes one or both of the terms in a sense that does not match the context induced by the corpus of text documents taken as a source of background knowledge. As previously pointed out (see previous section), an example is given by the relations *Table-subClassOf-Group* and *Step-subClassOf-Choice*.

In order to achieve the desired aim, the relation finding engine was modified as follows. As a first step the WordNet database was used to identify, for any given term, a set of so called *coordinate terms*, defined as the set of terms having a common hypernym in WordNet. The coordinate terms set is meant to contain semantically related terms. Clearly, based on the same term, different coordinate term sets will be obtained depending on which of the semantic senses is considered (i.e., depending on which WordNet synset is used for a given word). As an example, the set of coordinate terms for the word “group” is reported in Table 7.

In order to check which of the many possible senses of a certain word has to be taken into account by the relation discovering engine, a strategy was set up to measure the “degree of consistency” between a word sense and the context induced by the corpus of documents which are under exam. This context is modelled using the concept of Tag Cloud[3], which is common in the world of Web 2.0, and is used to collect a list of the *n* most used terms in a corpus, along with their relative frequency in the texts. The idea is that a Tag Cloud can give a representation of “what the corpus is about” and that the intersection between the coordinate terms set of a given term and the Tag Cloud “measures” to which extent the term itself matches a certain context.

Table 7. Three different senses for the word “group” (source WordNet), and their coordinate terms.

Sense	**Coordinate terms**
Any number of entities (members) considered as a unit	amount, measure, grouping, communication, set, relation, attribute, quantity, group, otherworld, psychological_feature
(Chemistry) Two or more atoms bound together as a single unit and forming part of a molecule	chemical_chain, chain, unit_cell, couple, molecule, group, radical, chemical_group
A set that is closed, associative, has an identity element and every element has an inverse	intersection, null_set, interval, range, range_of_a_function, root, topological_space, mathematical_space, image, solution, mathematical_group, field, subset, Mandelbrot_set, universal_set, domain, domain_of_a_function, diagonal, locus, group

In order to quantify the match, let T indicate the Tag Cloud set, with $f(s)$ indicating the relative weight of every term s in in T. Moreover, let $C(w)$ denote the set of coordinate terms for a given term w. In order to obtain a numeric measure of the consistency of the coordinate terms with the Tag Cloud, various strategies were attempted, as reported in Table 8, where m represents the measure.

[3] A tag cloud or word cloud (or weighted list in visual design) is a visual depiction of user-generated tags, or simply the word content of a site, typically used to describe the content of web sites. Tags are usually single words and are normally listed alphabetically, and the importance of a tag is shown with font size or color. Source: Wikipedia.

Table 8. Strategies to measure the extent to which a word "matches" a certain context.

Strategy 1	`Set m = 0`
	`For each word s in T`
	`        For each word in C(w)`
	`                If C(w) contains s`
	`                        m = m+1`
	`Set m = m/|C(w)|`

Strategy 2	`Set m = 0`
	`For each word s in T`
	`        For each word in C(w)`
	`                If C(w) contains s`
	`                        m = m+1+f(s)`
	`Set m = m/|C(w)|`

Strategy 3	`Set m = 1`
	`For each word s in T`
	`        For each word in C(w)`
	`                If C(w) contains s`
	`                        m = m*(1+f(s))`
	`Set m = m/|C(w)|`

While Strategy 1 simply measures the cardinality of the intersection between *T* and *C(w)*, the other two strategies try to adjust this measure for the frequency that a certain word has in the Tag Cloud: the higher the frequency, the more representative a word is in the Tag Cloud and in the text from which the Tag Cloud was built, the "more important" is the fact that *C(w)* contains the word. It is to be noted that "contains" is here to be interpreted in terms of string distances, as discussed in section 3.

Whereas in the example of section 4, for each term in the corpus, only its first WordNet synset was used for relation discovery, the synset of the term with the largest *m* score is used here. This potentially allows leaving out unwanted senses that may give rise to relations that are not of interest in a given domain (such as *Table-subClassOf-Group* and *Step-subClassOf-Choice* in the domain of tourism).

The updated relation discovery engine was implemented by a Java class named `TagCloudRelationExtractor`, and was run several times, with varying strategies and varying tag clouds. The best results were apparently obtained using Strategy 3, using a Tag Cloud that was built in order not to contain terms whose relative weight falls under 0.01.

Discovered relations are represented in Table 9.

Table 9. Relations identified by the `TagCloudRelationExtractor` after parsing a corpus of 88 reviews from http://www.accessatlast.com.

Term	**Relation**	**Term**	**Term**	**Relation**	**Term**
Barn	subClassOf	Building	Wheelchair	subClassOf	Chair
Cottage	subClassOf	Building	Studio	subClassOf	Apartment
Castle	subClassOf	Building	Wall	partOf	Building
Chalet	subClassOf	Building	Wall	partOf	Room
Chair	subClassOf	Furniture	Doorway	partOf	Wall
BookCase	subClassOf	Furniture	Wall	partOf	Hallway
Sofa	subClassOf	Furniture	Wall	partOf	Hall
Dresser	subClassOf	Furniture	Carport	subClassOf	Building
Bed	subClassOf	Furniture	Villa	subClassOf	Building
Suite	subClassOf	Apartment	Tub	partOf	Bathroom
Restaurant	subClassOf	Building	Wall	partOf	Room
Hospital	subClassOf	Building	Wall	partOf	Hallway
Stairs	subClassOf	Stairway	Garage	subClassOf	Building

While the number of identified relations is less than the one in Table 6, it seems that none of them presents inconsistencies with the context. Apparently, using this strategy, none of the senses for the words “Group” and “Choice” are found to be in line with the context and thus no relations involving the terms are exploited: as a consequence, the relations *Table-subClassOf-Group* and *Step-subClassOf-Choice* are left out of the result set.

6 Conclusions and future developments

The report described details about the design and implementation of an Ontology Evolution Manager that is able to integrate different engines to identify relations between terms. It then discusses an application of the system for establishing relations between terms in a corpus of text documents and those representing entities of a given ontology. The relation discovery engine illustrated in the example is based on WordNet and results coming from its application are encouraging, although the overall strategy can certainly be improved. Moreover, the example demonstrates that the architecture described in Gabbanini (2010) seems to offer a good support for the implementation of ontology evolution processes and is open for the integration of more refined strategies.

It is to be noted that a limit of the approach consists in the fact that WordNet has limited support for multiple word terms (for example terms such as “Indoor Space” cannot be found), so that not all relations involving concepts expressed by multiple words cannot be examined using the `WNRelationFinder` class alone. This limit can be overcome by the fact that the implementation of the relation finding process relies on the visitor pattern, which allows to apply a set of relation finding engines (each implementing the `IRelationFinder` interface) having different characteristics, in order to combine strengths of different source of background knowledge. In this perspective, it would be interesting to use OpenCyc (OpenCyc, 2010) to support the relation finding engine.

Section 5 introduced a technique for removing “spurious” relations: while it proved to be effective, it would be certainly useful to investigate more on procedures that automatically allow evaluating the “quality” of newly discovered relations, based on the context in which the system is operating. This is a relevant issue, discussed also in Zablith et al. (2010).

7 References

Burzagli, L., Como, A., Gabbanini, F., 2010. Towards the convergence of Web 2.0 and Semantic Web for e-Inclusion. In: Miesenberger, K., Klaus, J., Zagler, W., Karshmer, A. (Eds.), Computers Helping People with Special Needs. Vol. 6180 of Lecture Notes in Computer Science. Springer, pp. 343-350.

Gabbanini, F., 2010. On a Java based implementation of ontology evolution processes based on Natural Language Processing. Tech. Rep. 65-8, Institute for Applied Physics, Italian National Research Council.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Jaro, M. A., 1995. Probabilistic linkage of large public health data files (disc: P687-689). *Statistics in Medicine* 14:491–498.

JWI, 2010. Available at http://projects.csail.mit.edu/jwi/, last visited on 26/10/2010

OpenCyc, 2010. Available at http://www.opencyc.org/, last visited on 14/10/2010

Sabou, M., d'Aquin, M., Motta, E., 2008. Scarlet: Semantic relation discovery by harvesting online ontologies. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (Eds.), The Semantic Web: Research and Applications. Vol. 5021 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Berlin, Heidelberg, Ch. 72, pp. 854-858.

Sabou, M., d'Aquin, M., Motta, E., 2008b. Exploring the semantic web as background knowledge for ontology matching. In: Spaccapietra, S., Pan, J., Thiran, P., Halpin, T., Staab, S., Svatek, V., Shvaiko, P., Roddick, J. (Eds.), Journal on Data Semantics XI. Vol. 5383 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Berlin, Heidelberg, Ch. 6, pp. 156-190.

Scarlet, 2010. Available at http://scarlet.open.ac.uk/, last visited on 17/09/2010

SecondString, 2010. Available at http://secondstring.sourceforge.net/, last visited on 06/10/2010

Winkler, W. E., 1999. The state of record linkage and current research problems. Statistics of Income Division, Internal Revenue Service Publication R99/04. Available from http://www.census.gov/srd/www/byname.html last visited on 06/10/2010.

WordNet, 2010. Available at http://wordnet.princeton.edu/, last visited on 17/09/2010

Zablith, F., Sabou, M., d'Aquin, M., Motta, E., 2009. Ontology evolution with Evolva. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (Eds.), The Semantic Web: Research and Applications. Vol. 5554 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Berlin, Heidelberg, Ch. 80, pp. 908-912.

Zablith, F., d'Aquin, M., Sabou, M., Motta, E., 2010. Using ontological contexts to assess the relevance of statements in ontology evolution. In: Knowledge Engineering and Knowledge Management by the Masses. To appear in Lecture Notes in Computer Science. Springer-Verlag.