### Mesbah Khan

London, England | +44 7919 306 593

[LinkedIn](https://www.linkedin.com/in/mesbahkhan/) │ [khanm@ontoledgy.io](mailto:khanm@ontoledgy.io)

# Background

The task was to:

1. As input: 2 TSVs
2. Convert the above TSV files (triples) into an RDF/OWL model.
3. Find mappings between these two RDF models: semantic, syntactic, etc.. the methods
4. used are up to the candidate. They will get bonus points for creativity!
5. Generate (and save) a mapping file
6. Merge the two models using the created mappings
7. Write at least half a page on the approach taken, identifying problems and suggestions for improvement.

# Approach

## Algorithm

1. Analyse Triples
   1. Read in input triples into in-memory RDF triplestore structure (used this instead of RDF XML)
      1. Generate graphs
   2. population analysis
   3. report differences (syntactic) in a difference graph
2. Prepare mapping (semantic) – manual
3. Read in mapping triples
4. Inject mapping on difference graph
5. Inject difference graph into a merged graph

## Matching

There are several approaches to carry out ontology integration:

1. pure syntactic
   1. match on names
   2. match on properties
2. semantic
   1. Rational/Linguistic
   2. Empirical/Realist

## Implementation

#### Triplestore

I had a choice between python and golang as the two languages I am currently working with.

I choose golang primarily to explore the RDF libraries in golang.

Of the available RDF libraries, used github.com/wallix/triplestore as this was the easiest to perform simple graph analysis operations over the triples data.

#### Graph visualisation

Graphviz DOT language used to export analysis graphs as this was easy quick to work with. Given the small data size, performance was not an issue, and this was sufficient.

## Results

## Population Analysis

|  |  |  |  |
| --- | --- | --- | --- |
|  | tsv\_1 | tsv\_2 | merged |
| Number of Triples | 134 | 134 | 142 |
| Subjects | 37 | 37 | 39 |
| Predicates | 5 | 5 | 6 |
| Objects: | 81 | 81 | 83 |

## Matching Analysis

Matched Triples: 122

Triples in 1 not in 2: 12 (see Figure 1)

Triples in 2 not in 1:12 (see Figure 2)

There is a risk with this matching approach, as we know that the same triple in one source may have different interpretation or extension in the other source.

## Mapping Analysis

For the mismatched triples, the code produces two graph (.dot) files (see Figure 1 and 2) that need to be reviewed to create the mappings. In the current dataset, the two mismatched graphs reveal that the following classes share all the attributes and may be candidates for the mapping.

1. Congress -> Conference
2. JournalsorPeriodicals -> Journals

#### Basic (Linguistic/Rational) Semantic Mapping Approach

The simplest way to generate a mapping is to establish an appropriate semantic link for the two cases. The following mappings were added (see Figure 4) based on a simple rational analysis of the terms:

1. JournalsorPeriodcals sounds like a more general class; therefore, an rdf:subClassof link is added between them. (assuming JournalsorPeriodicals will be a broader class and all journals would be included in it)
2. Congress and Conference seem to be identical, and an owl:sameAs link has been added between them. (assuming they are similar events, and they share the same definitions in the data)

This analysis may need to be refined by looking at external definitions, e.g. IAPCO Meeting Industry Terminology publication considers the two type of events to be different (conferences are smaller-scale events than congresses!) (ref: <https://www.iccaworld.org/aeps/aeitem.cfm?aeid=909>)

# Further Improvements/Reflections

#### Code

This code could be further improved in several ways:

1. Use better structure to organise the packages based on the processing stages.
2. Move to object orientation and introduce types (golang equivalent of classes) and interfaces in each file, and possibly factories for object creation.
3. Add checks to report all pairs of classes that share all attributes
4. Currently, the triple processing does not separate literals from resources (all objects are being loaded as a resource), this could be added (however, it adds little value to the processing at this level)
5. Once the mapping is in place, the code could use inheritance processing to remove unnecessary remote links from subclasses to the properties shared with the parent class (i.e. subclasses should just inherit the properties from the parent class, and redundant remote links can be removed)
6. The code could also merge same-as objects into a new single object and have reference links back to the source system objects; this would make sense from an ontology perspective (one object in the ontology for one object in the real world).
7. Could use a graph database for visualisation and mapping analysis for more substantial volumes (probably neo4j)
8. Currently, the code does not use XML serialisation to produce any RDF XML output. This could be done but does not change anything in terms of the semantics of the statements.
9. The resource names could be presented in a more readable manner by stripping out the URL namespaces and replacing with short namespace codes

#### Mapping - Semantic (Realist/Empirical) Mapping Approach

An alternative, more reliable mapping process would be to ingest the instance data (i.e. records referring to individual books, journals etc.) from the two systems (represented by tsv1 and tsv2) and use this data to validate if the semantic assumptions at the schema level also hold at the data level. As the instance data was not available, this approach could not be explored/demonstrated.

As this is a mock example, it only contains a more straightforward (less realistic) case where the two similar objects share identical definitions. In real-world terminological datasets, definitions for similar objects are rarely identical. In those cases where the definitions are misaligned partially, analysis of the instance data is a powerful approach for disambiguation.

### CONTENTS AND COMPILATION INSTRUCTIONS

# Contents of zipping

1. rdf\_processor folder, containing:

code: all the code

inputs: contains the inputs and the mapping file

outputs: contains all the outputs of the code + renditions of the .dot graphs in png format.

Main.exe: executable.

Main.go: the main file that calls the orchestration function in the code.

1. This report

# Running the code without compiling

Execute main.exe from the root folder of the code after unzipping the package.

# Pre-requisites for compiling the code

1. Install go compiler (<https://golang.org/doc/install#install>)
2. Create go workspace folder
3. Setup $GOPATH environment variable and point to the Go folder.
4. Setup go workspace sub-folders (see <https://github.com/golang-standards/project-layout>)

The simplest option would be to set up the following folders:

$GOPATH\src

$GOPATH\bin

$GOPATH\pkg

1. Go libraries to get

Use “go get <address of library as written below>” on the command line after go is installed and GOPATH is set.

*github.com/wallix/triplestore*

*github.com/saintfish/chardet*

*github.com/TomOnTime/utfutil*

*github.com/emicklei/dot*

# Building and Running the Code

1. Unzip and copy the code in the $GOPATH\src folder
2. Run “go build main.go” in the code folder root
3. Run main.exe

# Outputs

1. The output folder should have newly created .dot files:
   1. graph\_1.dot (tsv\_1)
   2. graph\_2.dot (tsv\_2)
   3. extra\_in\_1.dot (triples in tsv\_1 not in tsv\_2)
   4. extra\_in\_2.dot (triples in 1 not in 2)
   5. merged\_difference\_graph (triples in c and d above merged)
   6. merged\_difference\_with\_mapping\_graph(e + mapping triples)
   7. merged\_graph\_including\_mapping (fully merged graph including mapping)
2. these .dot files can be viewed as text or visually rendered using gvedit.exe (available in Graphviz installer, <https://www.graphviz.org/download/>)

### Figures

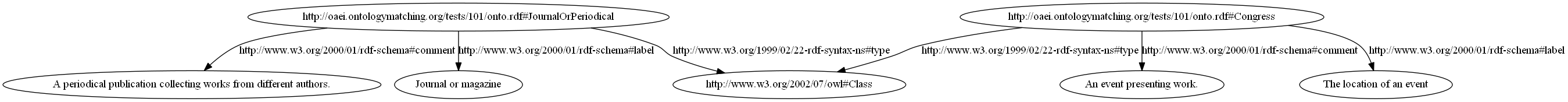


Figure 1 - Mismatched triples in 1.tsv not in 2.tsv

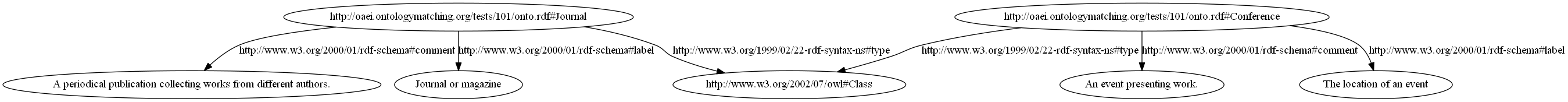


Figure 2 - Mismatched triples in 2.tsv not in 1.tsv

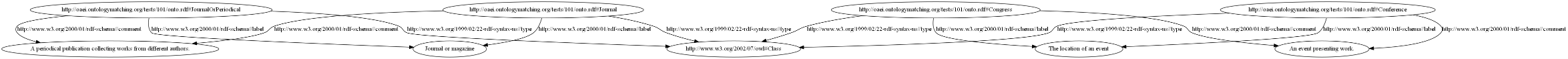


Figure 3 - Merged differences (no mapping)

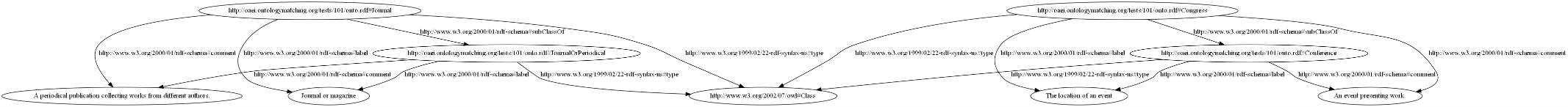


Figure 4. Merged differences (mapping)