YARRRML study

This document takes you through the steps of our study. The first section ('Preliminaries') takes you through the steps which should be carried out before session. The next section ('Process') explains the process for answering each question. The eight sections after that describe the eight questions, labelled '1' to '8'. Whilst undertaking these eight questions, we would like you to voice your reasoning to yourself as you answer the questions ("thinking aloud"); this gives us insight into your reasoning processes and any difficulties you encounter.

Preliminaries

Please undertake these steps before the session:

1. Create a folder to be used during the study. You can give this folder any name you like.
2. Download the JAR file, rmlmapper.jar, into your folder. This contains the code to map from RML to RDF. To download the file, go to:
   https://github.com/RMLio/rmlmapper-java
   Then click on 'latest', on the right of the page under 'Releases'. When you have downloaded the file, you will see that its name includes a reference to the version number. Rename the file to 'rmlmapper.jar'. This is in order that the file be correctly referenced from a macro which you will later define.
3. Download the zip file (experiment.zip) attached to the same email as this document. This zip file contains:
   a. eight files '1' to '8' with extension .yml
   b. eight files '1' to '8' with extension .ont
   c. five files containing data:
      i. artwork.json, which contains information, in JSON, about an artwork by the artist Robert Blake;
      ii. artwork.xml, which contains the same information as artwork.json, but in XML;
      iii. artworkAttributes.xml, which contains the same information as the previous two files; this file also uses XML, but unlike (ii) makes use of attributes;
      iv. artist38.csv, which contains CSV information about Robert Blake;
      v. artist_data.csv, which contains CSV information about five artists called 'Blake', including Robert Blake.
   d. a directory 'bak' which contains eight files '1' to '8' with extension .yml

Please unzip the experiment.zip file, so that all the files described in (3) above are in the folder you created in (1), with 'bak' a subfolder.

The files with extension .yml contain the YARRRML rules with certain lines incomplete. You will be asked to complete these lines during the study. The files with extension .ont contain the RDF ontologies which you are being asked to create. You can use these files to check that your completed rules have given the correct result. Also, if you are in any doubt about any question, you can consult the appropriate .ont file to better understand the required output. The 'bak' directory contains copies of the eight .yml files. You may not need to use these files; they are provided in case you wish to restart a question from scratch.

Before beginning the study, please carry out the following simple operations, which include defining a macro to execute the YARRRML mapping during the study.

For PC users

Open the command prompt, and use the 'cd' command to change your current working directory to the directory where you have stored the downloaded files. Then, to define the macro, cut and past the following line into your command prompt:

```
doskey y=yarrrml-parser -i $1.yml -o $1rml.ttl $T java -jar rmlmapper.jar -m $1rml.ttl -o $1.ttl
```

You will also need to download the YARRRML parser, i.e. the code which converts from YARRRML to RML. To do this, enter the following line into the command prompt (for this you will need Node.js installed):

```
npm i @rmlio/yarrrml-parser -g
```

This should be done with your current working directory unchanged, i.e. the parser should be in the same directory as the files extracted from experiment.zip.

For Mac users

Open the Mac terminal, and use the 'cd' command to change your current working directory to the directory where you have stored the downloaded files. Then, to define the macro, cut and paste the following lines into your command prompt:

```
function y () {
    yarrrml-parser -i $1.yml -o $1rml.ttl
    java -jar rmlmapper.jar -m $1rml.ttl -o $1.ttl
}
```

You will also need to download the YARRRML parser, i.e. the code which converts from YARRRML to RML. To do this, enter the following line into the command prompt (for this you will need Node.js installed):

```
npm i @rmlio/yarrrml-parser -g
```

This should be done with your current working directory unchanged, i.e. the parser should be in the same directory as the files extracted from experiment.zip.

Process

For each of the eight questions, you will be asked to complete a number of lines in a .yml file. To do this you can use any code editor, e.g. Notepad++, Atom etc. The text below states which lines are to be completed. The parts of these lines to be completed are identified by three dots (…).

You should then use the .yml file as input to the YARRRML mapping process. You can do this by using the macro 'y', which you have already defined. For example, to carry out question '1', which uses '1.yml' type:

```
y 1
```

This will create a file '1.ttl' which should contain the required ontology, i.e. it should be identical in content to '1.ont'. If this is not the case, then please re-edit the .yml file and resubmit the mapping. Where there is an error, the messages in the command prompt may be

helpful. In any case, a comparison between the .ttl and .ont files will help you to determine what changes need to be made. For each question, please continue until you have the correct output or until you feel unable to proceed. If necessary, when you are having difficulty, the experimenter will provide hints.

This process also creates a RML file, e.g. 1rml.ttl, 2rml.ttl etc. These files contain the RML which is generated from the YARRRML and which, in turn, generates the RDF. For the purposes of this study, there is no need to inspect these files.

Question 1

This makes use of two data files: artwork.json containing information about the artwork; and artist38.csv, containing information about the artist. Please open these two files, e.g. in Notepad++. You will need to inspect them to answer the question.

The objective of the question is to create:

- A triple with:
    - subject the URL of the page where the artwork can be viewed, this is on line 17 of the JSON file;
    - predicate 'dct:creator';
    - object the URL of the page describing the artist, which can be found in the CSV file.
- A set of five triples with subject the URL of the page describing the artist; and with predicates and objects describing the artist; see the last five lines of the 1.yml file for details. All this information comes from the CSV file.

Please open 1.yml and complete the seven lines indicated with … The first of these, line 13, is to identify the URL of the page where the artwork can be viewed; this URL can be found on line 17 of the JSON file. The URL forms the subject of a triple. The object of this triple is the URL of the page describing the artist, which must be taken from artist38.csv. To achieve this, please complete line 17. Finally, there are five lines (23 to 27) to be completed at the end of 1.yml which identify various pieces of information about the artist; all of this data is provided in artist38.csv.

Question 2

This also makes use of artwork.json, but this time in conjunction with artist_data.csv. The object is to create one triple, linking the URL of the page displaying the artwork (as used in question a) to the URL of the page describing the artist. The former URL is in the JSON file, the latter URL is in the CSV file. As the CSV file contains information about five artists, all with surname 'Blake', it is necessary to match the id for the artwork creator from the JSON file with that artist's id from the CSV file. Note that there are four lines to be completed: lines 12, 20, 21 and 25.

This question will give a warning message "mapping "artistMapping": no pos are defined." Please ignore this message.

Questions 3, 4 and 5

These questions all have the same objective. However, they use three different data files:

- artwork.json, a JSON file as used in the previous two questions;
- artwork.xml, an XML file which entirely uses XML tags and makes no use of attributes;
- artworkAttributes.xml, an XML file which makes use of attributes.

The three files are similar in organization, as far as that is possible allowing for the difference in data formats. Please note that artwork.json and artwork.xml use 'topics', i.e. in the plural, as a tag; however, consistent with the different approach to organizing the data, artworkAttributes.xml uses 'topic', i.e. in the singular.

The questions will be presented in different orders to different participants. However, it will be made clear which data file is to be used for each question, and in any case that will also be apparent from the YARRRML in the question.

In each file, a total of 13 descriptive topics are associated with the artwork. Each topic has a numeric id and a name, which is a character string containing a word or phrase. These topics are organized in a hierarchy, which is four levels deep. At the top level there is an over-arching topic with id = 1, name = 'topicRoot'. In this question, we are not concerned with this top-level topic, i.e. we are only concerned with the 12 topics in the three levels below the very top.

The objective of the question is to create two sets of triples:

- One set of triples with subject the URL of the page where the artwork can be viewed; with predicate 'schema:about' and with object an IRI generated from each of the topic ids by concatenating 'tsub:' and the id, e.g.
    - http://www.tate.org.uk/art/artworks/blake-two-drawings-of-frightened-figures-probably-for-the-approach-of-doom-a00002 schema:about tsub:273 .
- Another set of triples with subject an IRI representing each of the topics, i.e. as for the object of the previous triples; with predicate 'schema:name'; and with object the name of the topic, e.g.
    - tsub:273 schema:name "comforting" .

N.B. In the above examples the prefixes schema: and tsub: have been used for conciseness. In the ontology generated by the mapping process these prefixes will be expanded.

For each of these three questions, you need to complete six lines. The first three are to create the first set of triples described above, using 'artworkMapping':

- line 10 provides the iterator for the mapping;
- line 12 identifies the subject of these triples, i.e. the URL of the page where the artwork can be viewed;
- line 16 then identifies the topic ids, which are concatenated with tsub: to form the object of this first set of triples.

The remaining three lines to be completed are to create the second set of triples described above, using 'subjectMapping':

- line 20 provides the iterator required for the mapping;
- line 22 identifies the topic ids, to be concatenated with tsub:, to form the subject of these triples;
- line 26 identifies the name associated with the topic id, to form the object of this set of triples.

Questions 6, 7 and 8

These questions all have the same objective. As with the previous three questions, they differ in using the three different data files: artwork.json; artwork.xml; artworkAttributes.xml. Again, the order in which the data files are used varies between participants, but it will be clear at each stage what file to use.

In each case, the objective of the question is to create triples representing the hierarchy of topics describing the artwork. We represent topics in the same way as in previous questions, i.e. using IRIs generated from the topic id, e.g. tsub:273. In this question, we include the top-level topic, i.e. represented by tsub:1. We use 'skos:broader' as predicate to link each topic with each of its children, e.g.

- tsub:1 skos:broader tsub:29 .

Note that there will be 12 of these triples. Each topic IRI, except tsub:1, will occur once as the object of a triple.

The YARRRML solution for the JSON file is different than that for the two XML files. We discuss the JSON and XML questions separately below.

JSON file

For the question using the JSON file, there are two mappings in the YARRRML. The first mapping, 'lowerHierarchyMapping' creates the triples describing the lower levels of the hierarchy, i.e. all the triples with the exception of those whose subject is tsub:1. The second mapping, 'topHierarchyMapping' creates those triples whose subject is tsub:1.

In ‘lowerHierarchyMapping’, there are three lines to be completed:

- line 10 provides the iterator to be used for this mapping;
- line 12 then identifies the numeric topic id which, when concatenated with tsub:, forms the subject of a triple;
- line 16 identifies the numeric topic id which, concatenated with tsub:, forms the object of the triple.

The second mapping, ‘topHierarchyMapping’, is similar and also has three lines to be completed:

- line 20 provides the iterator for this mapping;
- line 22 then identifies the numeric topic id which, concatenated with tsub:, forms the subject of a triple;
- line 26 identifies the numeric topic id which, concatenated with tsub:, forms the object of the triple.

XML files

For the questions using the XML files, there is only one mapping: ‘hierarchyMapping’. For both questions, there are three lines to be completed:

- line 10 provides the iterator to be used for this mapping;
- line 12 then identifies the numeric topic id which, when concatenated with tsub:, forms the subject of a triple;
- line 16 identifies the numeric topic id which, concatenated with tsub:, forms the object of the triple.