Vrije Universiteit Brussel
Academic year 2020-2021
Group 1

Open Information Systems: milestone 4: Final report

Alexis Francois Verdoodt, Bram Dewit, Stijn Desloovere, Wolf De Wulf

(firstname.middlename.lastname@vub.be)[*]

Abstract

This document contains the documentation for our implementation of the project for the *Open Information Systems* course in the master of Computer Science at the Vrije Universiteit Brussel. The project consisted of designing and implementing an ontology and using the ontology to create an information system that is able to use its knowledge base in a non-trivial matter. The workload was divided perfectly equal as we always worked on the project simultaneously. The document is structured as follows. In the introduction we give a justification for our work and we explain the general ideas behind the ontology that we designed. Section 2 handles the conceptual schema and the database. In the third and fourth sections we describe the ontology itself and the mappings we devised to be able to link the database to the ontology and to allow us to set up a SPARQL endpoint. The fifth section explains what we did to demonstrate that our ontology can be used to perform non-trivial tasks. We conclude in the sixth section with a link to the project's github repository and a brief guide on how to install and use the system.

1 Introduction

We were allowed to choose from two general subjects for our ontology: video conferencing and travelling. We chose the latter. At the moment, due to the pandemic that is raging throughout the world, travelling activity has diminished substantially. With the idea of creating something useful, we chose to focus on the influence of pandemics on travelling rather than on travelling itself. We designed an ontology that models geographical places and associates them with their personal instances of numbers that are generally analyzed during pandemics. An example of such a number is the number of total infections. Not only do we model numbers, we also represent restrictions and advice that places issue to contain pandemics. The connection to travelling is that before thinking about vacation, one might want to know how the place they are thinking about going to is doing pandemic wise. Once a person knows what the state of the place is, we provide possible ways of getting there in the form of connections.

We created a database containing data on the covid-19 pandemic and *lifted* its conceptual schema with respect to the ontology. To demonstrate how the ontology can be used to perform particular tasks, we created a simple script that combines the database and the ontology into an information system that allows users to request covid-19 data for specific places, request both general and personal travelling advice and request the global standings pandemic wise.

[*]Student numbers shown by Table 1.

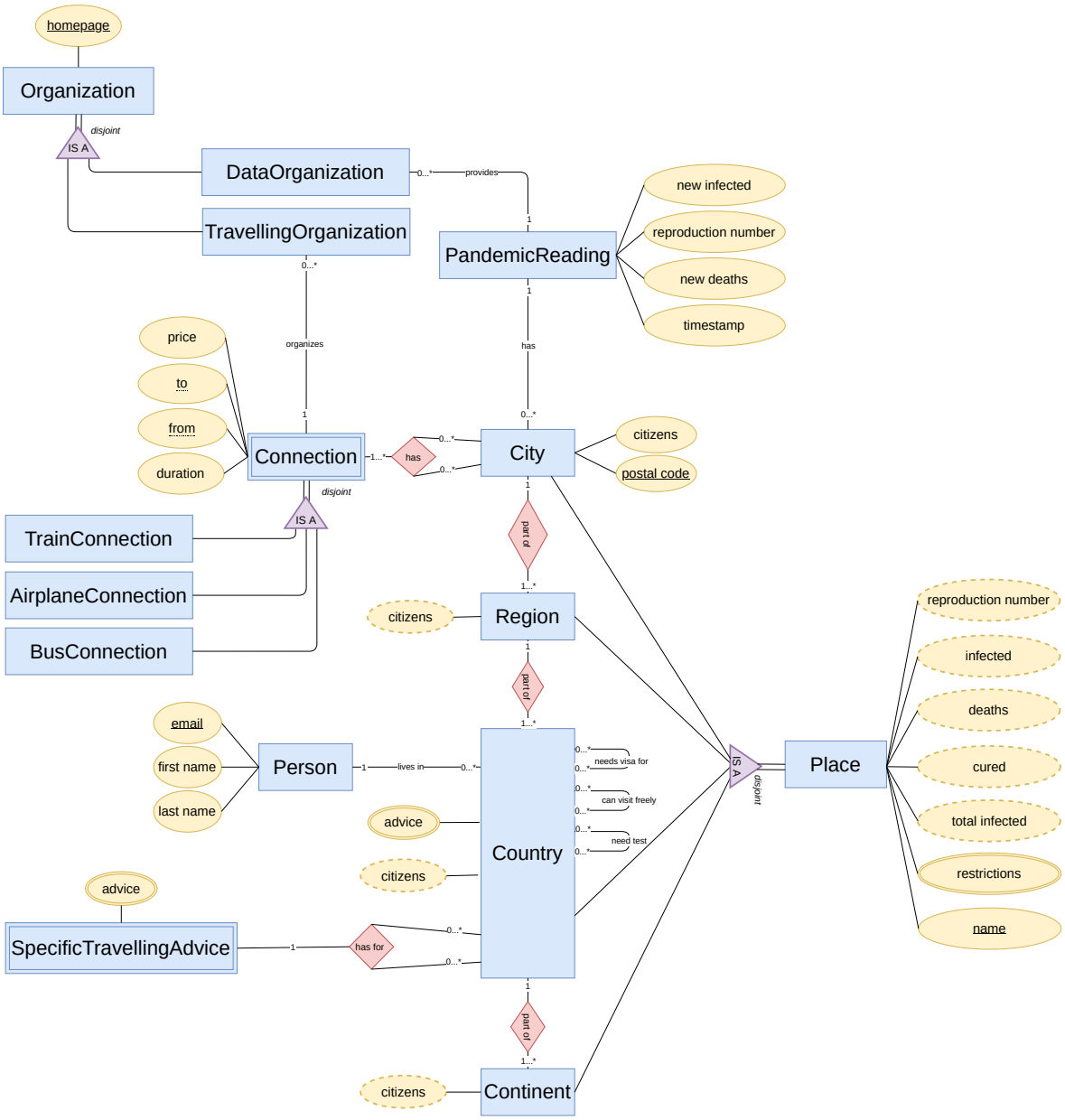

Figure 1: Entity relationship diagram for the H2 database.

2 Database

To ensure a good integration with the *Ontop Protégé* plugin we chose to use an H2[1] database. The package we provided contains an initialized database and hence can be used immediately. Figure 1 depicts the ER model for the database. Throughout this document it will become clear that the general concept that we are handling can semantically be divided into four parts. From now on, explaining the modules of the total package will always be done by giving explanations for the four separate parts.

Places Our database stores four tiers of places: cities, regions, countries and continents. For each of these we store two non-numeric fields that contain the restrictions the place issued and the place name. Places also have multiple derivable fields that represent data regarding the pandemic the system handles. For the three upper tiers, these derivable fields can all be calculated from the lowest tier: cities. In their turn, cities can calculate their data from the zeroth tier: readings. Readings store a timestamp indicating at what moment its observations were done, as well as values for the actual observations: new infections, new deaths, new cured cases and a reproduction number. Note that instead of implementing views in our H2 database that calculate the derivable fields for the higher tier places, we opted for assigning this task to the mappings. If the database were to be used with an application that does not make use of the SPARQL endpoint, implementing views would be necessary.

Organizations There are two types of organizations: one that organizes connections between cities and one that provides readings. Identifying organizations is done by their websites.

Connections There are three types of connections: train connections, bus connections and airplane connections. Note that we store connections only on the city tier.

Advice For each country we store travelling advice that the government of the country has issued with respect to the pandemic the system handles. With the idea of creating a minimal viable product we also store persons. For each of them we store their full name, e-mail address and their country of residence. On top of general travelling advice we also store specific travelling advice. These consist of advice countries have issued for a specific other country.

3 Ontology

Figure 2 shows a WebVOWL[2] representation of the ontology we designed. As you can see, the structure is very similar to that of the ER model. In what follows we explain the same four parts from section 2 but now with respect to the ontology instead of the ER model. Where it applies we also declare the parts we adopted from external ontologies.

Places Creating new axioms for geographical places is something one should not do, as there are multiple existing ontologies that already have axioms for these subjects. For our four tiers of geographical places we adopted the *dbo:Continent*, *dbo:country*, *dbo:region* and *dbo:city* classes from the DBpedia[3] ontology. DBpedia itself defines these classes to be subclasses of *dbo:Place*, which we also adopted. In the graphical representation you can see that we extended *dbo:Place* with data properties that model the earlier mentioned pandemic analysis numbers. To be able to calculate these numbers in a real instance of the ontology, there need to be data readings of course. Said readings are represented by our own *PandemicReading* class which is connected to *dbo:city* using an object property called *reading*. Lastly, we extended *dbo:country* with three object properties that are self-loops. They represent the following relations countries can have with other countries: inhabitants of a country need a visa to enter a specific other country, inhabitants of a country need a negative test for the disease the ontology handles to enter a specific other country and inhabitants of a country can freely enter a specific other country.

Organizations For organizations we were able to import existing axioms as well. From the FOAF[4] ontology, we imported *foaf:Organization* and subclassed it with our own *DataOrganization* and *TravellingOrganization*. To model that the data inside a *PandemicReading* is provided by a *DataOrganization* we created

[1] https://www.h2database.com/html/main.html
[2] http://vowl.visualdataweb.org/webvowl.html
[3] https://wiki.dbpedia.org/
[4] http://xmlns.com/foaf/spec/

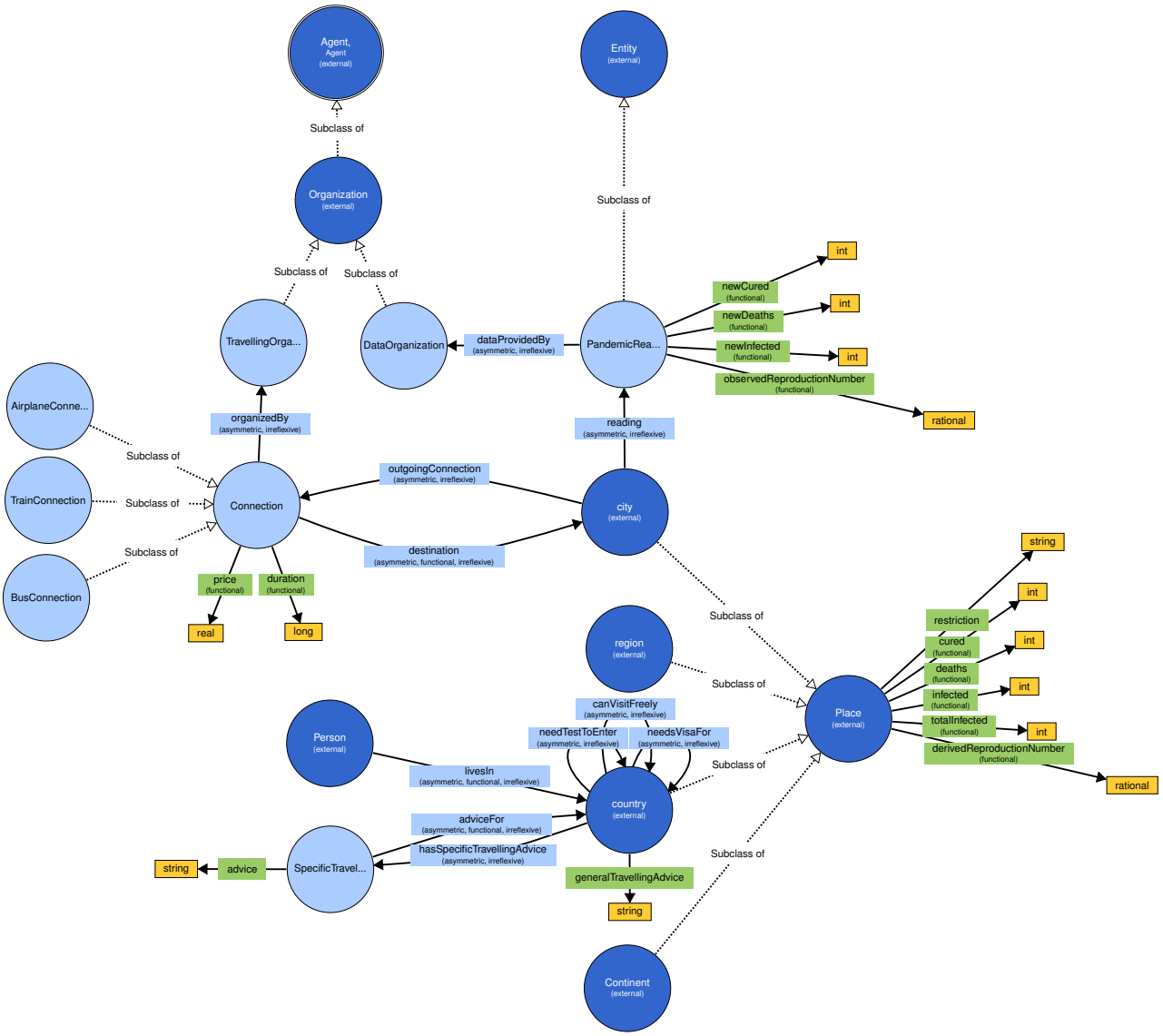

Figure 2: WebVOWL representation of the ontology.

our own object property *dataProvidedBy* and made it a sub property of PROV-O's[5] *prov:wasAttributedTo*. To make sure that the domain and range of our new sub property match that of its super class, we made *PandemicReading* a subclass of *prov:Entity* and wrote an axiom that states that *foaf:Agent*, the superclass of *foaf:Organization*, is an equivalent of *prov:Agent*.

Connections For this part of the ontology we did not import any external axioms. We created a class *Connection* that is connected to *dbo:city* via an object property *desination*. In the opposite direction, *dbo:city* is connected to *Connection* via an object property *outgoingConnection*. A connection has two data properties *price* and *duration* and it also has three subclasses: *BusConnection*, *TrainConnection* and *AirplaneConnection*. Lastly, connections are provided by a *TravellingOrganization*. This is modeled through the object property *organizedBy*.

Advice General travelling advice is modeled by the *generalTravellingAdvice* data property that we added to *dbo:country*. On the other hand, specific travelling advice can not be modeled by a data property, which is why we created a *SpecificTravellingAdvice* class that is connected to *dbo:country* through the object property *adviceFor*. In the opposite direction, *dbo:country* is connected to *SpecificTravellingAdvice* through the object property *hasSpecificTravelligAdvice*. Note that this pattern matches that of the object properties between *Connection* and *dbo:city*. Lastly, to model users we imported *foaf:Person* and made a new *livesIn* object property to connect a person to *dbo:country*.

4 Mappings

In this section we introduce the mappings that map entries in the database to triples for a SPARQL endpoint triplestore. To create the mappings we made use of the Ontop[6] plugin for Protégé[7]. This means that we did not use the R2RML mappings we were taught in the course, but rather the native Ontop mapping language, which is a bit easier to learn and use. Again, we explain the mappings for each part separately, note that we will not go into each and every mapping but rather focus on the non-trivial ones.

Places For geographical places, the most complicated mappings are those that calculate the analysis numbers for each place, using the place's connections to the lower tier places. Mapping the database entries for *PandemicReading*s to triples that represent the analysis numbers for a *dbo:city* is done by the following mapping:

```
mappingId
    CityDerivable
target
    :data/City/{cityName} :infected {infected}^^xsd:int ;
    :totalInfected {totalInfected}^^xsd:int ;
    :cured {cured}^^xsd:int ;
    :deaths {deaths}^^xsd:int ;
    :derivedReproductionNumber {reproductionNumber}^^owl:rational .
source
    SELECT
        cityName,
        SUM(newInfected) - SUM(newCured) as infected,
        SUM(newInfected) as totalInfected,
        SUM(newCured) as cured, SUM(newDeaths) as deaths,
        AVG(reproductionNumber) as reproductionNumber
    FROM PandemicReading GROUP BY forCity
```

We declare the target triples using the data properties we defined for the analysis numbers, basically creating empty triples that need to be filled with data. Said data is retrieved from the database using the given SQL query. By doing a join on the *partOf* fields and grouping correctly we can easily calculate the values of the analysis numbers for the higher tier places:

[5] https://www.w3.org/TR/prov-o/
[6] https://protegewiki.stanford.edu/wiki/Ontop
[7] https://protege.stanford.edu/

```
mappingId
    RegionDerivable
target
    :data/Region/{regionName} :Infected {infected}^^xsd:int ;
    :totalInfected {totalInfected}^^xsd:int ;
    :cured {cured}^^xsd:int ;
    :deaths {deaths}^^xsd:int ;
    :derivedReproductionNumber {reproductionNumber}^^owl:rational .
source
    SELECT
        regionName,
        SUM(newInfected) - SUM(newCured) as infected,
        SUM(newInfected) as totalInfected,
        SUM(newCured) as cured, SUM(newDeaths) as deaths,
        AVG(reproductionNumber) as reproductionNumber
FROM Region r
     inner join City c on r.regionName = c.partOf
 inner join PandemicReading cr on c.cityName = cr.forCity
GROUP BY r.regionName
```

Note that this means that per tier, an extra join is needed. Luckily, these are the only cases for which joins are ever needed. All of the other mappings are straightforward and need no joins at all. We therefore omit the three other parts of the system in this section as the mappings for them are fairly simple and need no explanation. We would like to forward the interested reader to the mappings themselves.

5 Demonstrator

We created a simple application to demonstrate that our ontology can be used to perform a non-trivial task. The application consists of a simple C++ script. The script focuses on the advice and places parts of our ontology. When booted, users can choose from four options: enter their country of residence, request covid-19 data for a specific place, request travelling advice or request the global standings for total infected or total deaths. When choosing the advice option, the output depends on if the user has entered their country of residence. If so, the advice that the script returns will be specific travelling advice based on their country of residence, if not it will be general travelling advice.

The script consists of a very simple, infinite loop that keeps giving the same four options. Behind the screens, the script queries the SPARQL endpoint using template queries that it fills with the user's input.

6 The system

A public github repository that contains all the files related to this project can be found at: `https://github.com/wulfdewolf/OpenInformationSystemsGroup1`. The `README.md` file contains a manual on how to set-up each part of the project as well as a guide on how to use them. We made use of the *Docker* framework to make it easy for anyone to get started with the system. Once *Docker* is installed, a single command is all that is needed to boot the complete system. The demonstrator is not included in the docker containers, however, executing its binary inside a terminal requires minimal effort.

The ontology file can be found at `endpoint/ontology/ontology.owl`, the mappings file bears the same name but uses the `.obda` file extension.

Name	Student number
Alexis Francois Verdoodt	0545813
Bram Dewit	0545676
Stijn Desloovere	0545142
Wolf De Wulf	0546395

Table 1: Table that contains the group member's student numbers.