Development of FactForge

15.03.2013

1. Introduction

FactForge represents a reason-able view over several important Linked Open Data (LOD) datasets. It enables users to easily identify resources in the LOD cloud by providing a general unified method for querying a whole group of datasets. FactForge is designed also as a use-case for large-scale reasoning and data integration. This document describes the development of its March 2013 release - datasets, ontologies, inference rules, and manipulations done over the data.

In brief, the datasets are unified via a common ontology – PROTON, which concepts are mapped to the concepts of the involved LOD datasets. We do this by a set of rules. Each of them maps a PROTON class or a PROTON property to the corresponding class or property of the other ontologies. This mechanism of constructing a reason-able view over selected LOD datasets ensures that the redundant instance representations (classes and properties) are cleaned as much as possible. The instances are grouped in equivalent classes of instances. Finally, the instances in these datasets are linked via owl:sameAs statements.

FactForge development can be divided into six main steps:

1. Selecting the LOD datasets
2. Checking each dataset for consistency
3. Mapping the PROTON concepts to the respective LOD datasets concepts
4. Cleaning the datasets from any discrepancies between the concepts in the different datasets and PROTON
5. Loading all datasets in a joint repository
6. Loading owl:sameAs statements and checking for consistency

Here, we also present solutions for resolving discrepancies when mapping concepts from the central datasets in LOD (also loaded in FactForge) and PROTON, as well as the way of cleaning the datasets.

In some of the cases, we have to add new instances, which are introduced via inference rules.

Ultimately, FactForge provides a deeper understanding of: the Linked Open Data available on the web, some peculiarities of the datasets conceptualization and the problems of integrating the different LOD datasets.

2. Mapping rules

This section describes the methodology for creating a correspondence between two dataset conceptualizations of the real world. When constructing such correspondence, several manipulations of the datasets facts are conducted: (1) introducing new individuals; (2) deleting some individuals; (3) modifying some individuals; (4) inserting/deleting/updating relations between individuals; (5) inserting/deleting/updating characteristics of the individuals. The idea behind LOD is that such transformations are minimal. Ideally, there should be no transformations at all. We respect this recommendation, as much as possible,

when constructing FactForge, except in such cases where the resulting reason-able view contradicts the conceptualization of the PROTON ontology.

Therefore, in developing FactForge, our first aim is to support a full querying of the resulting repository via PROTON classes and properties. This means that we allow classes and properties from PROTON to be part of the formulation of SPARQL queries. We use only rdfs:subClassOf or rdfs:subPropertyOf statements in order to ensure a complete mapping coverage of the PROTON ontology to the other schemas in FactForge. Generally the mapping statements can be arbitrary couples but in most cases they are simply rdfs:subClassOf or rdfs:subPropertyOf statements between classes or properties explicitly defined in the PROTON ontology and the ontology or the schema of a given dataset. For example[1]:

dbp:SportsTeam rdfs:subClassOf pext:Team .

foaf:homepage rdfs:subPropertyOf pext:hasWebPage .

However, due to the different conceptualizations, in some cases a more complex mapping is needed. For example, in the Geonames dataset geographical objects are classified by codes and not by an ontology hierarchy. In such cases the mapping is done by more complex statements such as:

[rdf:type owl:Restriction ;

owl:onProperty <http://www.geonames.org/ontology#featureCode> ;

owl:hasValue <http://www.geonames.org/ontology#A.PCL>]

rdfs:subClassOf pext:Country .

Some of these compound statements require adding new individuals. In such cases, we use the OWLIM inference rules to create the necessary additions. Here is an example:

//dbp-ont:PrimeMinister rdfs:subPropertyOf [ptop:hasPosition [pupp:hasTitle]].

Id:PM

p <rdf:type> <dbp-ont:PrimeMinister>

---

[1] Here are the namespace declarations used in the document:

@prefix ptop: <http://www.ontotext.com/proton/protontop#> .
@prefix pext: <http://www.ontotext.com/proton/protonext#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dbp: <http://DBpedia.org/ontology/> .
@prefix dbp-prop: <http://DBpedia.org/property/> .
@prefix fb: <http://rdf.freebase.com/ns/>
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

p <ptop:hasPosition> j

j <pext:hasTitle> <pext:PrimeMinister

Here, the inference rule is necessary because the conceptualizations in the DBpedia ontology and in the PROTON ontology are different. In DBpedia, Prime Minister belongs to a class of politicians, which is a class of person, while in PROTON, Prime Minister is a title of a job position. Therefore, in DBpedia, the respective Prime Minister is an individual whereas in PROTON he is an individual who has a position with the title PrimeMinister. As the instance data about the position itself (j in the rule above) is missing in the DBpedia dataset, it has to be created so that the mapping between the two ontologies is consistent.

3. DBpedia Ontology and Dataset

The DBpedia dataset is created by extracting structured information from Wikipedia and presenting it in an RDF form (http://DBpedia.org/About). The conceptualization of the DBpedia dataset is based on the categories that are designed and implemented in Wikipedia, i.e. the data in the info-box section of the articles. This conceptualization is presented as ontology. For our purposes, we have used version 3.8. It contains 359 classes, 800 object properties and 975 data types. In addition, some of the classes and properties of these other ontologies are used in the definition of the DBpedia ontology. The instances in the DBpedia dataset are classified according to the conceptual information in its ontology, but also to some other ontologies - http://schema.org; http://xmlns.com/foaf/spec/. Reuse of classes and properties from one in the definition of another ontology is considered desirable behavior having in mind the cost of creating ontologies. On the other hand, providing the mapping between ontologies could cause problems, as we will see below. Also using classes and properties from ontologies that are not explicitly used in the dataset is often useless. Thus, we have to clean the dataset from unwanted statements. In the majority of cases, the conceptualizations of the DBpedia and PROTON ontologies are compatible and the mapping between them is straightforward as discussed earlier. However, there are still some differences as illustrated in the following two examples:

Architect as a Person

In the DBpedia ontology, many roles in society, mainly performed by persons, are formalized as subclasses of the class dbp-ont:Person.

dbp-ont:Architect

rdf:type owl:Class;

rdfs:subClassOf dbp-ont:Person .

The definition in PROTON is:

pext:Architect

rdf:type pext:Profession ;

rdfs:comment "A profession of planning, design and oversight

of the construction of buildings and some other

artefacts. (Wikipedia)"@en .

and

pext:Profession

rdf:type owl:Class ;

rdfs:subClassOf pext:SocialFunction .

The main difference is that in PROTON the class pext:Architect is defined as a profession and a social function, in order for someone (or something) to have this profession. This means that not only persons can perform it. While in DBpedia the definition follows the logic that all architects described in Wikipedia are, in fact, persons.

It is relatively easy to overcome such conceptual differences by an appropriate mapping between the two ontologies:

dbp-ont:Architect rdfs:subClassOf [rdf:type owl:Restriction ;

owl:onProperty pext:hasProfession ;

owl:hasValue pext:Architect] .

This statement determines that all instances of dbp-ont:Architect correspond to the instances of the PROTON ontology with the profession pext:Architect.

Sport as an Activity

Another example is the definition of Sport. In the DBpedia ontology, it is defined as:

dbp-ont:Sport

rdf:type owl:Class;

rdfs:comment "A sport is commonly defined as an organized, competitive,

and skillful physical activity."@en;

rdfs:subClassOf dbp-ont:Activity .

and in PROTON as:

pext:Sport

rdf:type owl:Class ;

rdfs:comment "A specific type of sport game"@en ;

rdfs:subClassOf pext:SocialAbstraction .

The difference is that in DBpedia, Sport is a specific activity and its characteristics such as game rules, number of participants, etc. are not defined in the class dbp-ont:Sport. In PROTON the characteristics of the sport game are defined in the class pext:Sport as a social abstraction. The actual realization of the definition as a sport event is an instance of Activity.

Unfortunately, any mapping between the two ontologies cannot solve this conceptual difference. The following mappings:

dbp-ont:Activity

rdfs:subClassOf pext:Activity .

and

dbp-ont:Sport

rdfs:subClassOf pext:Sport .

automatically make all instances of the class dbp-ont:Sport in PROTON to be simultaneously instances of the classes ptop:Happening and ptop:Abstract, which are mutually disjoint.

In FactForge such conceptualization differences between the two ontologies are solved by not loading the DBpedia ontology in the FactForge repository. In this way, we make use of the richness of the DBpedia instance data but impose the conceptualization of the PROTON ontology over it.

Another reason for not loading the DBpedia ontology is that the definitions in it also contain mappings to other ontologies. However, we believe that including ontology statements referring to classes (properties, etc) of other ontologies is not a good practice. First, presenting the necessary conceptualization requires importing the other ontology. And second, this can introduce some contradictions in the ontology that is using these statements.

For example, the DBpedia ontology contains some statements from the Schema ontology (http://schema.org), which take care of the mapping between this ontology and the DBpedia ontology. However, because DBpedia is not an extension of the Schema ontology, therefore it is better to store these statements separately. If they are included in the definitions of the DBpedia classes, this can lead to some contradictions as illustrated in the examples below:

dbp-ont:University

rdf:type owl:Class;

rdfs:subClassOf dbp-ont:EducationalInstitution ;

owl:equivalentClass schema:CollegeOrUniversity .

and

```
dbp-ont:College a owl:Class;
    rdf:type owl:Class;
    rdfs:subClassOf dbp-ont:EducationalInstitution ;
    owl:equivalentClass schema:CollegeOrUniversity .
```

Using owl:equivalentClass makes these two classes - dbp-ont:University and dbp-ont:College - the same. Such equivalent statements are difficult to be noticed in the DBpedia ontology as it is full of, them but, it is also not very easy to use DBpedia without such statements.

The instance data also contains statements that result from inferences from the DBpedia ontology. In order to avoid all conceptualizations that follow from the DBpedia ontology we have to clean the DBpedia instance data from such inferences. Here are some examples:

Subclass - Superclass inference

In the DBpedia instance data, each instance of sport is classified as sport but also as an activity. Therefore, even if we do not load the DBpedia ontology in the FactForge repository, this inference is present in the instance data. Thus, the classification of the DBpedia sport instances will also be wrong in PROTON when mapping PROTON to DBpedia.

To clean this instance data statement we have created a deletion statement of the following type:

```
delete {?s a dbp-ont:SuperClass} where
            {
              ?s a dbp-ont:SubClass .
              ?s a dbp-ont:SuperCLass .
            }
```

Here is an example:

```
delete {?s a dbp-ont:Activity} where
            {
             ?s a dbp-ont:Sport .
             ?s a dbp-ont:Activity .
            }
```

In this way, if there is a statement for a subclass, we delete all the statements for the super classes. After that, we use the inference mechanisms of the repository to make the inferences that follow from the mapping to the PROTON ontology.

rdfs:domain and rdfs:range statements

In the DBpedia instance data, some statements for domain and range have properties connected to instances that do not belong to the appropriate classes. Such unclassified instances in DBpedia could be wrongly classified in PROTON, based on these domain and range statements. In order to clean such cases we use queries of the following type:

```
delete {?s dbp-ont:DBpediaProperty ?y } where

{?s dbp-ont:DBpediaProperty ?y .

 ?y rdf:type ?c .

filter(

        ?c = dbp-ont:Class01

    ||  ?c = dbp-ont:Class02

    ||  ... ## List of all unappropriate classes

    )

}
```

Here is an example of the property dbp-ont:birthPlace.

```
delete {?s dbp-ont:birthPlace ?y } where
{?s dbp-ont:birthPlace ?y .
 ?y rdf:type ?c .
filter(?c = dbp-ont:AcademicJournal    ||   ?c = dbp-ont:Activity
   ||   ?c = dbp-ont:AdministrativeRegion    ||   ?c = dbp-ont:Aircraft
   ||   ?c = dbp-ont:Airline    ||   ?c = dbp-ont:Airport
   ||   ?c = dbp-ont:Album    ||   ?c = dbp-ont:AmericanFootballLeague
   ||   ?c = dbp-ont:AmericanFootballTeam    ||   ?c = dbp-ont:Amphibian
   ||   ?c = dbp-ont:AnatomicalStructure    ||   ?c = dbp-ont:Animal
   ||   ?c = dbp-ont:Arachnid    ||   ?c = dbp-ont:Archaea
   ||   ?c = dbp-ont:ArchitecturalStructure    ||   ?c = dbp-ont:Arena
   ||   ?c = dbp-ont:Artery    ||   ?c = dbp-ont:Artwork
   ||   ?c = dbp-ont:Asteroid    ||   ?c = dbp-ont:Atoll
   ||   ?c = dbp-ont:AustralianFootballLeague    ||   ?c = dbp-ont:AutoRacingLeague
   ||   ?c = dbp-ont:Automobile    ||   ?c = dbp-ont:AutomobileEngine
   ||   ?c = dbp-ont:Award    ||   ?c = dbp-ont:Bacteria
   ||   ?c = dbp-ont:Band    ||   ?c = dbp-ont:BaseballLeague
   ||   ?c = dbp-ont:BaseballTeam    ||   ?c = dbp-ont:BasketballLeague
   ||   ?c = dbp-ont:BasketballTeam    ||   ?c = dbp-ont:Beverage
   ||   ?c = dbp-ont:BiologicalDatabase    ||   ?c = dbp-ont:Biomolecule
   ||   ?c = dbp-ont:Bird    ||   ?c = dbp-ont:BodyOfWater
   ||   ?c = dbp-ont:Bone    ||   ?c = dbp-ont:Book
   ||   ?c = dbp-ont:BowlingLeague    ||   ?c = dbp-ont:BoxingLeague
```

|| ?c = dbp-ont:Brain || ?c = dbp-ont:Bridge
|| ?c = dbp-ont:BroadcastNetwork || ?c = dbp-ont:Broadcaster
|| ?c = dbp-ont:Building || ?c = dbp-ont:CanadianFootballLeague
|| ?c = dbp-ont:CanadianFootballTeam || ?c = dbp-ont:Canal
|| ?c = dbp-ont:Cave || ?c = dbp-ont:CelestialBody
|| ?c = dbp-ont:ChemicalCompound || ?c = dbp-ont:ChemicalElement
|| ?c = dbp-ont:ChemicalSubstance || ?c = dbp-ont:Church
|| ?c = dbp-ont:City || ?c = dbp-ont:ClubMoss
|| ?c = dbp-ont:College || ?c = dbp-ont:Colour
|| ?c = dbp-ont:ComicBook || ?c = dbp-ont:Company
|| ?c = dbp-ont:Conifer || ?c = dbp-ont:Constellation
|| ?c = dbp-ont:Continent || ?c = dbp-ont:Convention
|| ?c = dbp-ont:Country || ?c = dbp-ont:CricketLeague
|| ?c = dbp-ont:Crustacean || ?c = dbp-ont:CurlingLeague
|| ?c = dbp-ont:Currency || ?c = dbp-ont:Cycad
|| ?c = dbp-ont:CyclingLeague || ?c = dbp-ont:Database
|| ?c = dbp-ont:Decoration || ?c = dbp-ont:Device
|| ?c = dbp-ont:Disease || ?c = dbp-ont:Drug
|| ?c = dbp-ont:EducationalInstitution || ?c = dbp-ont:Election
|| ?c = dbp-ont:Embryology || ?c = dbp-ont:EthnicGroup
|| ?c = dbp-ont:Eukaryote || ?c = dbp-ont:EurovisionSongContestEntry
|| ?c = dbp-ont:Event || ?c = dbp-ont:Fern
|| ?c = dbp-ont:FieldHockeyLeague || ?c = dbp-ont:Film
|| ?c = dbp-ont:FilmFestival || ?c = dbp-ont:Fish
|| ?c = dbp-ont:Flag || ?c = dbp-ont:FloweringPlant
|| ?c = dbp-ont:Food || ?c = dbp-ont:FootballMatch
|| ?c = dbp-ont:FormulaOneRacing || ?c = dbp-ont:Fungus
|| ?c = dbp-ont:Galaxy || ?c = dbp-ont:Game
|| ?c = dbp-ont:Gene || ?c = dbp-ont:GeneLocation
|| ?c = dbp-ont:GeopoliticalOrganisation || ?c = dbp-ont:Ginkgo
|| ?c = dbp-ont:GivenName || ?c = dbp-ont:Gnetophytes
|| ?c = dbp-ont:GolfLeague || ?c = dbp-ont:GovernmentAgency
|| ?c = dbp-ont:GovernmentType || ?c = dbp-ont:GrandPrix
|| ?c = dbp-ont:Grape || ?c = dbp-ont:GreenAlga
|| ?c = dbp-ont:HandballLeague || ?c = dbp-ont:HistoricBuilding
|| ?c = dbp-ont:HistoricPlace || ?c = dbp-ont:HockeyTeam
|| ?c = dbp-ont:Holiday || ?c = dbp-ont:Hospital
|| ?c = dbp-ont:Hotel || ?c = dbp-ont:HumanGene
|| ?c = dbp-ont:HumanGeneLocation || ?c = dbp-ont:IceHockeyLeague
|| ?c = dbp-ont:Ideology || ?c = dbp-ont:Infrastructure
|| ?c = dbp-ont:InlineHockeyLeague || ?c = dbp-ont:Insect
|| ?c = dbp-ont:Instrument || ?c = dbp-ont:Island
|| ?c = dbp-ont:LacrosseLeague || ?c = dbp-ont:Lake
|| ?c = dbp-ont:Language || ?c = dbp-ont:LaunchPad
|| ?c = dbp-ont:LawFirm || ?c = dbp-ont:LegalCase
|| ?c = dbp-ont:Legislature || ?c = dbp-ont:Letter
|| ?c = dbp-ont:Library || ?c = dbp-ont:Lighthouse
|| ?c = dbp-ont:Locomotive || ?c = dbp-ont:LunarCrater
|| ?c = dbp-ont:Lymph || ?c = dbp-ont:Magazine
|| ?c = dbp-ont:Mammal || ?c = dbp-ont:MeanOfTransportation
|| ?c = dbp-ont:MilitaryConflict || ?c = dbp-ont:MilitaryUnit
|| ?c = dbp-ont:Mineral || ?c = dbp-ont:MixedMartialArtsEvent
|| ?c = dbp-ont:MixedMartialArtsLeague || ?c = dbp-ont:Mollusca
|| ?c = dbp-ont:Monument || ?c = dbp-ont:Moss
|| ?c = dbp-ont:MotorcycleRacingLeague || ?c = dbp-ont:Mountain
|| ?c = dbp-ont:MountainPass || ?c = dbp-ont:MountainRange
|| ?c = dbp-ont:MouseGene || ?c = dbp-ont:MouseGeneLocation
|| ?c = dbp-ont:Muscle || ?c = dbp-ont:Museum
|| ?c = dbp-ont:MusicFestival || ?c = dbp-ont:MusicGenre
|| ?c = dbp-ont:Musical || ?c = dbp-ont:MusicalWork

```
  || ?c = dbp-ont:Name || ?c = dbp-ont:NationalSoccerClub
  || ?c = dbp-ont:NaturalPlace || ?c = dbp-ont:Nerve
  || ?c = dbp-ont:Newspaper || ?c = dbp-ont:Non-ProfitOrganisation
  || ?c = dbp-ont:OlympicResult || ?c = dbp-ont:Olympics
  || ?c = dbp-ont:Organisation || ?c = dbp-ont:PaintballLeague
  || ?c = dbp-ont:Painting || ?c = dbp-ont:Park
  || ?c = dbp-ont:PeriodicalLiterature || ?c = dbp-ont:Place
  || ?c = dbp-ont:Planet || ?c = dbp-ont:Plant
  || ?c = dbp-ont:Play || ?c = dbp-ont:PoliticalParty
  || ?c = dbp-ont:PoloLeague || ?c = dbp-ont:PopulatedPlace
  || ?c = dbp-ont:PowerStation || ?c = dbp-ont:ProgrammingLanguage
  || ?c = dbp-ont:Project || ?c = dbp-ont:ProtectedArea
  || ?c = dbp-ont:Protein || ?c = dbp-ont:PublicTransitSystem
  || ?c = dbp-ont:Race || ?c = dbp-ont:RadioControlledRacingLeague
  || ?c = dbp-ont:RadioStation || ?c = dbp-ont:RailwayLine
  || ?c = dbp-ont:RailwayTunnel || ?c = dbp-ont:RecordLabel
  || ?c = dbp-ont:ReligiousBuilding || ?c = dbp-ont:Reptile
  || ?c = dbp-ont:ResearchProject || ?c = dbp-ont:Restaurant
  || ?c = dbp-ont:River || ?c = dbp-ont:Road
  || ?c = dbp-ont:RoadJunction || ?c = dbp-ont:RoadTunnel
  || ?c = dbp-ont:Rocket || ?c = dbp-ont:RouteOfTransportation
  || ?c = dbp-ont:RugbyClub || ?c = dbp-ont:RugbyLeague
  || ?c = dbp-ont:Sales || ?c = dbp-ont:SambaSchool
  || ?c = dbp-ont:School || ?c = dbp-ont:Sculpture
  || ?c = dbp-ont:Settlement || ?c = dbp-ont:Ship
  || ?c = dbp-ont:ShoppingMall || ?c = dbp-ont:Single
  || ?c = dbp-ont:SiteOfSpecialScientificInterest || ?c = dbp-ont:SkiArea
  || ?c = dbp-ont:Skyscraper || ?c = dbp-ont:SnookerWorldRanking
  || ?c = dbp-ont:SoccerClub || ?c = dbp-ont:SoccerClubSeason
  || ?c = dbp-ont:SoccerLeague || ?c = dbp-ont:SoccerLeagueSeason
  || ?c = dbp-ont:SoccerTournament || ?c = dbp-ont:SoftballLeague
  || ?c = dbp-ont:Software || ?c = dbp-ont:Song
  || ?c = dbp-ont:SpaceMission || ?c = dbp-ont:SpaceShuttle
  || ?c = dbp-ont:SpaceStation || ?c = dbp-ont:Spacecraft
  || ?c = dbp-ont:Species || ?c = dbp-ont:SpeedwayLeague
  || ?c = dbp-ont:SpeedwayTeam || ?c = dbp-ont:Sport
  || ?c = dbp-ont:SportsEvent || ?c = dbp-ont:SportsLeague
  || ?c = dbp-ont:SportsTeam || ?c = dbp-ont:SportsTeamSeason
  || ?c = dbp-ont:Stadium || ?c = dbp-ont:Station
  || ?c = dbp-ont:Stream || ?c = dbp-ont:SupremeCourtOfTheUnitedStatesCase
  || ?c = dbp-ont:Surname || ?c = dbp-ont:Tax
  || ?c = dbp-ont:TelevisionEpisode || ?c = dbp-ont:TelevisionSeason
  || ?c = dbp-ont:TelevisionShow || ?c = dbp-ont:TelevisionStation
  || ?c = dbp-ont:TennisLeague || ?c = dbp-ont:TennisTournament
  || ?c = dbp-ont:Theatre || ?c = dbp-ont:TopicalConcept
  || ?c = dbp-ont:Town || ?c = dbp-ont:TradeUnion
  || ?c = dbp-ont:Tunnel || ?c = dbp-ont:University
  || ?c = dbp-ont:Unknown || ?c = dbp-ont:Valley
  || ?c = dbp-ont:Vein || ?c = dbp-ont:VideoGame
  || ?c = dbp-ont:VideogamesLeague || ?c = dbp-ont:Village
  || ?c = dbp-ont:Volcano || ?c = dbp-ont:VolleyballLeague
  || ?c = dbp-ont:WaterwayTunnel || ?c = dbp-ont:Weapon
  || ?c = dbp-ont:Website || ?c = dbp-ont:WineRegion
  || ?c = dbp-ont:WomensTennisAssociationTournament || ?c = dbp-ont:Work
  || ?c = dbp-ont:WorldHeritageSite || ?c = dbp-ont:WrestlingEvent
  || ?c = dbp-ont:WrittenWork || ?c = dbp-ont:Year || ?c = dbp-ont:YearInSpaceflight )
}
```

Apart from the deleted statements discussed earlier, we have deleted all instance data described by statements using classes that are not from the DBpedia ontology.

In this way, the DBpedia instance data has a clean interpretation in terms of the PROTON conceptualization.

4. Freebase dataset

Freebase (http://www.freebase.com/) is a community-curated database of well-known people, places, and things. In Freebase, real-world entities are represented as topics. For example, there are topics for movie stars, countries, cities, etc. The information for each topic is structured in three levels as defined in the Freebase schema. The first layer comprises several domains (76). Each domain is defined by type (second layer) and each type has properties (third layer). All topics are described as belonging to one or more types via the property http://rdf.freebase.com/ns/type.object.type

The types are connected via the special relation *inclusion of type*. This relation connects more specific types with more general types. For example, the type https://www.freebase.com/base/litcentral/named_person includes the type: https://www.freebase.com/people/person. However, it is not possible to interpret this relation as superclass-to-subclass relation, because it is not strict in the sense that each instance of the subclass inherits the properties of the instance of the super class. For example, the type https://www.freebase.com/film/actor also includes the type https://www.freebase.com/people/person. But its definition is: "The Film Actor type includes people (and credited animals) who have appeared in any film ...". Therefore, in most cases, the instances of the type https://www.freebase.com/film/actor are people but there are also cases where they are not. Consequently, the interpretation of the type inclusion relation is not strict with respect to inheritance of the properties from the included type. In the example above, if the film actor is a person, then he or she inherits all properties from the type for persons. But if it is not a person, then it does not inherit any of these properties. Instead, it inherits properties from some other type(s).

These peculiarities of the Freebase schema impose some restrictions over the mapping to the PROTON ontology. Mapping so many types and properties requires more extensive work. Therefore, for our purposes, we have mapped only the types with more than 500 instances in the Freebase dataset to the PROTON concepts. Another criterion is that the mapping does not produce any misclassification of some instances. For many types, the mapping is straightforward:

```
fb:location.location

        rdf:type owl:Class;

        rdfs:comment "The Location type is used for

                any topic with a fixed location..."@en ;

        rdfs:label "Location";

    rdfs:subClassOf ptop:Location .
```

For types representing professions and other social roles, the mappings are similar to the mapping used for the DBpedia ontology:

```
fb:military-militarycommander

        rdfs:subClassOf

                [rdf:type owl:Restriction ;

                 owl:onProperty pext:hasTitle ;

                 owl:allValuesFrom pext:Commander].
```

Some of the types are mediators between a type and a grouping of several other types. This is mainly used to represents event information. For example, the type *Website ownership* describes an event of owning a website by an agent for some period. A website can be owned by different agents in different periods, thus it is important that these 'owning' events are represented as different instances in the dataset.

At present, we have not yet mapped the mediator types to PROTON. For this type of mapping it is necessary to use an appropriate subclass of the class ptop:Happening. For example, the type *Website ownership* can be mapped to a subclass of the class ptop:Situation, where the start and end date of the ownership are stated, the owner and the address of the website are specified, etc. As this requires huge extension of PROTON, it is not featured in the current version.

In the original dataset, there are also several errors in the instance classification. For example, organisation and location are very often represented by the same instance. More specifically, the types fb:location.location and fb:organization.organization have 42763 instances in common. We believe that such cases result from the linguistic intuition of the users who created the data in question. In many cases, the same word denotes both the meaning of an institution and a location. We do not consider this a good practice for semantic representation in LOD and we think that it should be avoided. The different classes (types in Freebase) have different properties. Although, the Freebase types are not strict in inheriting properties, some types are still not mutually compatible (intuitively). For example, due to this misclassification, the instance of the United State of America (https://www.freebase.com/m/09c7w0) is not only an instance of the types Country, Location but also of Food. Therefore, we believe that such knowledge has to be represented in a different way altogether.

It is important to note that correcting such cases of classification of instances to many disjoint types (classes) is outside the scope of FactForge.

5. Conclusion

This document presents the main problems of using together two of the most popular LOD datasets: DBpedia and Freebase. The main lessons learned are as follows:

1. The world can be modelled in many different ways, which can be formally incompatible but still understandable by human users. It is true that the main value of a dataset is in its usefulness to the users. However, this is not enough in terms of the Semantic Web where the goal is to have LOD datasets that can be processed by machines. To achieve this, it is necessary to apply some formal evaluation of the represented knowledge.

2. The incompatibility can appear on different levels: granularity of conceptualization, representation of different kinds of knowledge (for example, the difference between sortals and roles), etc. Generally, the conclusion is that if we want LOD to achieve their goals, they should not only follow some formats but also their conceptualizations should adhere to certain restrictions to ensure compatibility.
3. Constructing new ontologies based on existing ones has to incorporate the complete semantics of the corresponding ontologies instead of just fragments of them. Such an approach will have an effect on the consistency of the new ontologies and their interoperability with the existing ones.