Things to be careful about when developing an ontology in RDF

Many domains

When you need a property for several classes, instead of having a property with several domains (which is wrong; it means intersection and not union), it's advisable to create a new class, which is superclass of all those classes and then the domain of the property would be the new superclass. In this way, the property is inherited to all classes you want to.

E.g. both "*Employees*" and "*Customers*" have "*names*". But we don't create the "*name*" property having both classes as a domain, because this means that only objects that belong to BOTH classes (intersection) can have names. Instead we should create a class "*Person*" as a superclass of "*Employee*" and "*Customer*" and use that new class as the domain of property "*name*".

Many ranges

When a property can have as a range many alternative classes, do not put them all in the range, because they are considered as intersection (AND) and not as union (OR). Instead, find a common superclass of all these classes and put it as a range. If these classes do not have a common superclass, then either create one (if it makes sense), or leave the range blank, or put *rdfs:Resource* as a range. You cannot specialize the range for some of the classes that inherit that property in RDF.

E.g. Students attend either postgraduate or undergraduate courses. But we should not put as the range of the "*attends*" property both classes "*PostGraduateCourse*" and "*UnderGraduateCourse*", because this would mean that students attend courses that are both "*PostGraduateCourse*" and "*UnderGraduateCourse*" (i.e. belong to the intersection of these classes), which is wrong. Instead, we should create class "*Course*" as a superclass of "*PostGraduateCourse*" and "*UnderGraduateCourse*" and use that class as the range of the "*attends*" property. Of course, this class always exists.

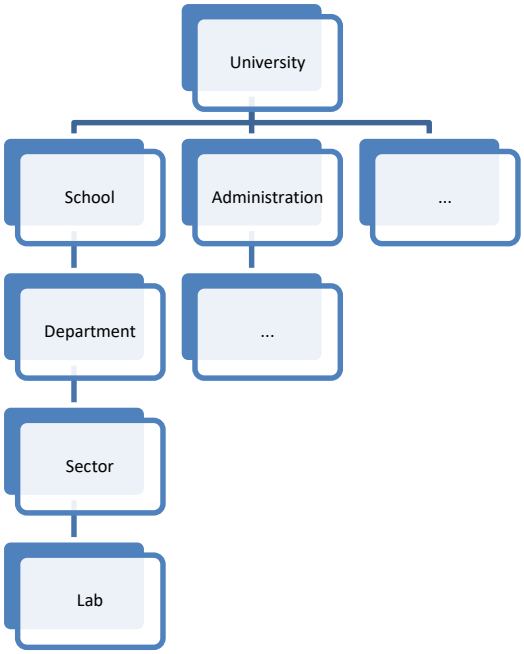
There are cases though, that the developer had not thought before of a common superclass. This "need" in the ontology would lead him/her to create such a common superclass. E.g. imagine that we model our company and its accounting office. The accounting office makes transactions by buying goods or services. Each transaction has a date, an amount of money and the beneficiary (the one who will get the money). The beneficiary is either a supplier or a courier company. So the range of the "beneficiary" property needs both these classes. Instead of putting them both (WRONG!) we should create a new class, e.g. *BusinessPartner*, and put these two classes as subclasses and *BusinessPartner* would be the range of the "beneficiary" property.

IMPORTANT NOTICE FOR PROTÉGÉ-OWL: The above discussion is only valid when you manually edit an RDF file. The user-interface of Protégé does not allow you to enter multiple classes in the domain and the range of a property, when using the "Pure RDF Schema" profile.

Part-of instead of is-a hierarchy

Don't try to overdo it with class hierarchies. A height 3-4 would usually suffice. Don't try to put all of your classes into a single class hierarchy tree; this usually leads to mistakes. You could have multiple class

independent class hierarchies. E.g. a common mistake you should avoid is that sometimes people confuse the hierarchical relationship *part-of* or *belongs-to*, with the relationship *is-a*. The class hierarchy represents an "*is-a*" relation: a class *A* is a subclass of *B* if every instance of *A* is also an instance of *B*. E.g. a postgraduate student *is-a* student, a lecturer *is-a* faculty member, etc. An example of mistaking this is the following:



In the above example, a lab belongs to a sector, a sector belongs to a department, a department belongs to a school, and a school belongs to a university, so there is a hierarchical relationship among these concepts/classes, however this is NOT the "*is-a*" relationship, but the "*part-of*" or "*belongs-to*" relationship. Therefore, it is completely wrong to model the above with *rdfs*:*subclassOf*.

Instead, each class should be modeled independently of each other and they can be linked through a relationship called "*belongsTo*". Probably this property is a transitive one, but RDF/S lacks this feature.

Sub-properties

The same mistake with the class hierarchy is usually done for the property hierarchy. Sometimes the value of a property has inherently a complex structure, e.g. *dateOfBirth*, which consists of day, month and year. If you want to keep this piece of information separately, then you need three different properties: *dayOfBirth*, *monthOfBirth* and *yearOfBirth*. All these together constitute the *dateOfBirth*, so each one of them is "*part-of*" *dateOfBirth*. This cannot be modeled with *rdfs*:*subPropertyOf*; it is wrong! Actually, this relationship cannot be modeled at all, unless you create a new class, called *DateOfBirth*. Instances of this class will be the birthdates of all the people in your ontology. Properties of this class would be *dayOfBirth*, *monthOfBirth* and *yearOfBirth*. Each person will be linked to his/her birthdate with an *rdf*:*Property*, e.g. *hasBirthDate*, whose domain would be class *Person* and range class *DateOfBirth*. So, if you want the values of properties to be complex objects, instead of atomic values, then they should be modeled as objects.

The real meaning of the relationship *rdfs*:*subPropertyOf* is that whenever the sub-property holds between two resources, then the same is true for the super-property, but not the other way around. Usually the

relationship *rdfs*:*subPropertyOf* is between object properties (e.g. *father->parent*, *isTaughtBy->involves*, etc.), but this is not always the case. You could have a sub-property relationship with datatype properties, as well. E.g. each person can have a home phone, a work phone, a mobile phone. All of them are contact phones. So, the property "*phone*" can be a super-property of "*home_phone*", "*work_phone*", and "*mobile_phone*". All of these properties are datatype properties.

Naming classes and properties

Classes are usually named with a capital first letter and they should be in singular. E.g. class "*Student*" and not "*student*", "*students*", "*Students*".

Properties are usually named with a lower case first letter, e.g. "*age*", "*isTaughtBy*".

More details

More details about how to develop an ontology can be found at:

Natalya F. Noy and Deborah L. McGuinness. ``Ontology Development 101: A Guide to Creating Your First Ontology''. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

http://www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html