1-1-2013

Providing metrics and automatic enhancement for hierarchical taxonomies

Ghassan Beydoun
University of Wollongong, beydoun@uow.edu.au

Francisco Garcia-Sanchez
University of Murcia

Cristin M. Vincent-Torres
University of Murcia

Antonio A. Lopez-Lorca
University of Wollongong, aall645@uowmail.edu.au

Rodrigo martinez-Bejar
University of Murcia

Providing metrics and automatic enhancement for hierarchical taxonomies

Abstract

Taxonomies enable organising information in a human-machine understandable form, but constructing them for reuse and maintainability remains difficult. The paper presents a formal underpinning to provide quality metrics for a taxonomy under development. It proposes a methodology for semi-automatic building of maintainable taxonomies and outlines key features of the knowledge engineering context where the metrics and methodology are most suitable. The strength of the approach presented is that it is applied during the actual construction of the taxonomy. Users provide terms to describe different domain elements, as well as their attributes, and methodology uses metrics to assess the quality of this input. Changes according to given quality constraints are then proposed during the actual development of the taxonomy. (C) 2012 Elsevier Ltd. All rights reserved.

Keywords

hierarchical, enhancement, automatic, metrics, taxonomies, providing

Disciplines

Engineering | Science and Technology Studies

Publication Details

Beydoun, G., Garcia-Sanchez, F., Vincent-Torres, C. M., Lopez-Lorca, A. A. & martinez-Bejar, R. (2013). Providing metrics and automatic enhancement for hierarchical taxonomies. Information Processing & Management, 49 (1), 67-82.

Providing Metrics and Automatic Enhancement for Hierarchical Taxonomies

Abstract Taxonomies enable organising information in a human-machine understandable form, but constructing them for reuse and maintainability remains difficult. The paper presents a formal underpinning to provide quality metrics for a taxonomy under development. It proposes a methodology for semi-automatic building of maintainable taxonomies and outlines key features of the knowledge engineering context where the metrics and methodology are most suitable. The strength of the approach presented is that it is applied during the actual construction of the taxonomy. Users provide terms to describe different domain elements, as well as their attributes, and methodology uses metrics to assess the quality of this input. Changes according to given quality constraints are then proposed during the actual development of the taxonomy.

Keywords: Incremental knowledge development, taxonomies, ontology evaluation, data models, knowledge monitoring.

1 INTRODUCTION

Taxonomies are a restricted form of domain specific knowledge where relations between domain specific entities are all of the form *Is-A*. They have been important for modelling database schemas, knowledge-based systems and semantic vocabularies (Guarino and Welty 2001). They often come from different organisations and persons with varying agendas with varying quality criteria. Interest in their evaluation in the context of their design within semantically enabled technologies increased in recent years. Some examples of this can be found in (Middleton 2004) where a similarity system as a prelude to evaluation is presented, or in (Staab et al. 2001), where the authors propose a complex framework of various characteristics across content, language, development methodology, building tools, and usage cost. The need for a taxonomy evaluation is identified in three scenarios (Borst, 1997): where both users and machines need a taxonomy assessment guide to suit their needs; where designers need practical guides to build and evaluate taxonomies before publishing them, or where automatic taxonomy machine-learning requires identifying a suitable option among varying different possibilities, to adjust the parameters of the learning algorithms appropriately. Although there are some examples of evaluation approaches in literature which accommodate the psychological/mental/real processes that take place in create taxonomies, very few of them provide a formal structure (Brewster et al. 2004). This paper presents a formalization that is well suited in all three scenarios. More importantly, it presents an evaluation approach which targets the scenario where designers are helped during the evolution of the taxonomy and before it is delivered. We present a

well-founded evaluation framework for automatic structural assessment of hierarchical taxonomies by means of a set of formalised metrics and sketch how it can be deployed in the context of the knowledge models development. We present and illustrate an algorithmic methodology for building taxonomies with formally specified content.

The rest of the paper is organised as follows: Section 2 presents related. Section 3 presents the formalization to enable description of taxonomies precisely. Section 4 uses the formalization to describe quality criteria and uses these to create concomitant evaluation and correction algorithms. Section 5 presents an illustration how the algorithms can be used and outlines how the work can be integrated in an incremental knowledge development process. Section 6 concludes with a discussion of future extensions to this work.

2 RELATED WORK

Taxonomies are envisioned to be developed by domain experts having limited or no skills in knowledge engineering. Therefore, it is important to provide supporting and appropriate techniques and tools to enable their effective development and evaluation. To this end, evaluation of knowledge models generally have been discussed in the literature for more than twenty years, e.g. (Grogono et al. 1991). *We identify two categories of* evaluation approaches, domain dependent and domain independent approaches. The existing domain dependent approaches provide formal metrics e.g. (Menzies 1998; Brewster et al. 2004). Menzies (1998) provides a domain dependent statistical assessment of a classification system. This can only be applied only during or after deployment of the system. Brewster et al. (2004) define conditions related to concept boundaries based on philosophical issues such as identity, unity, essence, dependency, and rigidity. The domain dependent approaches identify structural features of the knowledge base but can only be used once a knowledge model is sufficiently mature e.g. (Ning and Shihan 2006; Punera et al. 2006). For example, Punera et al. (2006) provide a characterization of what a "good" knowledge model is once it is completed rather than during its construction as we propose to be done in this work. Software engineering goals checking techniques, after the development of the taxonomies have also been proposed in (Nick et al. 1999; Sadrei et al. 2007). These target knowledge management goals in an organisation, and they are of little use in classification tasks as in taxonomies where there is exactly one goal, that of classification. Our work is distinguished from previous work that it targets at evaluation during construction of the taxonomy model. It is most suited in settings where the human expert is still present during the development. Moreover, we assume that the expert providing the expertise is not necessarily aware of knowledge engineering issues and this lack of awareness will be compensated through our automatic support. This paper provides quantitative measures to evaluate taxonomies and uses these to support a domain expert improves his knowledge engineering outcome, the taxonomy.

The formalization of taxonomies presented in this paper is not completely new, for example, a system that checks taxonomical relationships in background, giving information about possible inconsistencies is presented in (Voelker et al. 2005). Another system to describe the structure of an ontology so that a developer can make decisions to improve it is presented in (Ning and Shihan 2006). However, in our approach, once a problem is spotted, our approach also provides support for re-arrangements of relationships involving the ill-represented concepts. We apply a formalization similar in scope to the one used in

(Guarino and Welty 2004) which analyzes a taxonomic ontology by using meta-properties representing intended meaning of classes, properties and relations, and its purpose is to obtain the correct taxonomic category for each class in the ontology. Our formally specified algorithm to fix malformed taxonomies is also valuable in settings where taxonomies are developed on the fly by non-computer literate users e.g. collecting user preferences (Chamiel and Pagnucco 2008; Balke and Wagner 2004). Well formed taxonomic structure are a key technology in Semantic Web (Chamiel and Pagnucco 2008) where every concept in the hierarchy (not only leaves) can be referred to, not only as an abstract concept (probably through the set of properties it contains), but also as a *data concept*— a concept which can be associated with asserted instances (i.e., real web objects). A hierarchical musical genre system is of this kind: an album can be identified as *Progressive Rock* which is a leaf concept but at the same time an album can be identified with the concept *Rock* even though it serves as an abstract concept. Various other similar examples can be found in the literature e.g. in (Schickel-Zuber and Faltings 2006; Beydoun et al. 2011; Fortuna et al 2007).

3 MODELLING MAINTAINABLE TAXONOMIES

In this section, we present our taxonomy formalisation structured around concepts, IS-A relations, and axioms. The election of terms is incrementally user-guided, so it ends up being closer to the users' wishes, making resultant taxonomies easier to use. Our emphasis is on mathematical modelling to build (well-based) taxonomies. We use set theoretic definitions to formally describe the components (concepts, attributes and their values) of a taxonomy and put these together to formally define a taxonomy. We first introduce definitions 1 and 2 to formalise attributes requirements:

Definition 1: **Well-defined attribute:** Let *A* be an attribute of some concept *c* so that *V* stands for the alphabet used to form terms in a certain domain, *n* represents the maximum length allowed for any term, and *M* is a user-definable set among *N, Z, Q,* and *R. A* is said to be well-defined, written *well-defined(c,a),* if there exists a function, written *POSSIBLE_VALUES (c, A),* defined as $V^n \times V^n \rightarrow F$ and such that the following conditions hold:

i) $c \neq A$
ii) *Cardinal(F)* ≥ 1 *where Cardinal(F) refers to the number of elements in F*
iii) $\{c\} \cap POSSIBLE\_VALUES\ (c, A) = \phi$
iv) $\{a\} \cap POSSIBLE\_VALUES\ (c, A) = \phi$

The range of values an attribute can have depends on the level of abstraction associated with the definition of the attribute. This leads to Definition 2:

Definition 2: **Attribute abstraction:** Let *a* be a well-defined attribute of a concept *c*. Then, the level of abstraction of *a*, *a_abstraction (c,a)*, is defined as follows:

$$\text{a\_abstraction}(c,a) = \begin{cases} d & \text{if } F \in \{\mathcal{N}, \mathcal{Z}, Q\} \\ D & \text{if } F \in \{\mathcal{R}, C\} \\ \text{Cardinal (F) otherwise} \end{cases}$$

where F is defined according to Definition 1 and holding that Cardinal(F) < d < D;
F $\in$ *{N,Z} means that F has countable infinite elements; and*

$F \in \{Q,R\}$ means that F has uncountable infinite elements.

For well-defined attribute, *a*, of a concept *c*, *a* is a constant attribute, written if *a_abstraction (c,a) = 1*. Otherwise, it is said to be an abstract attribute.

To formalise notions used to describe individual concepts, definitions 3 to 5 are introduced:

Definition 3: **Conceptual internal structure/well-defined concept:** Let *c* be a concept; let a_i be a well-defined attribute for *c*; let *V* be the alphabet used to form both terms *a* and *c*; and let *n* be the maximum length allowed for the terms. The internal structure of *c*, *IN(c)*, is defined as $V^n \rightarrow P(V^n)$ with $IN(c) = \{ a_i | \ well\_defined\ (c, a_i)\}$ and *c* is said to be a well-defined concept.

Definition 4: **Concept abstraction:** Let *c* be a well-defined concept and let *IN(c)* be its internal structure. Then, the level of abstraction of *c*, written *c_abstraction(c)*, is defined as the vector $[a\_abstraction\ (c, a_1), \ldots, a\_abstraction\ (c, a_m)]$ where a_i is i-th element of *IN(c)* in *Cardinal(IN(c))*.

Note for a concept *c*, *c* is said to be an instance if $\forall x \in IN(c)$, $a\_abstraction\ (c, x) = 1$. On the other hand, *c* is said to be an abstract (concept), if $\exists\, x \in IN(c)$, $a\_abstraction\ (c,x) > 1$.

Definition 5: **Completely/incompletely defined instance:** Let *c* be an instance with *IN(c)* being its internal structure. *c* is said to be incompletely defined, written *incomplete_instance(c)*, if the following conditions hold:

i) $\exists x \in IN(c)$ s.t. its value is precise though unknown (at the moment of modelling). Then, we say that $POSSIBLE\_VALUES(c,x) = \{unknown\}$

ii) $Cardinal(\{x \text{ s.t. } POSSIBLE\_VALUES(c,x) = \{unknown\}) < Cardinal(IN(c))$

It is natural to use (sub)categories in understanding or explaining a new concept. Hence, we present in a step-by-step process our definition of concepts from this category-centered point of view. To deal with categories of concepts, definitions 6 and 7 are introduced:

Definition 6: **Upper/lower level category:** Let *c* and *c'* be two different well-defined concepts, with *IN(c)* and *IN(c')* being their respective internal structures. *c'* is said to be an upper level category of *c*, written *upper(c',c)*, if all of the following conditions hold:

i) $IN(c') \subseteq IN(c)$
ii) $\forall x \in IN(c')$, $a\_abstraction(c,x) \leq a\_abstraction(c',x)$
iii) $\exists y \in IN(c')$ s.t. $a\_abstraction(c,y) < a\_abstraction(c',y)$

Definition 7: **Upper/lower level conceptual context:** Let C be a non-empty set of well-defined concepts. The upper level conceptual context of a given concept $c \in C$, written *upper_context(c,C)*, is defined as $\{c' \in C \text{ s.t. } upper(c',c)\}$. Similarly, the lower level conceptual context of a given concept $c \in C$, written *lower_context(c,C)*, is defined as $\{c' \in C \text{ s.t. } lower(c',c)\}$.

Example 1: Illustrating definitions 1 to 7

In the set of well-defined concepts *C={vehicle, bicycle, car, truck}*, the concept "*bicycle*" has the following attributes *{number_of_wheels, capacity_of_load, trademark, number_of_passengers}* The attributes of the concept "*truck*" are *{number_of_wheels, capacity_of_load, trademark}*.

POSSIBLE_VALUES(bicycle, number_of_wheels)=2
POSSIBLE VALUES(bicycle, capacity_of_load)=200 Kg
POSSIBLE_VALUES(bicycle, trademark)= {Any string}
POSSIBLE_VALUES(bicycle, number_of_passengers)={1,2}
POSSIBLE_VALUES(truck, number_of_wheels)={4,6,8}
POSSIBLE VALUES(truck, capacity_of_load)=100000 Kg
POSSIBLE_VALUES(truck, trademark)= {Any string}

The concept "*car*" would be well-defined if we assume that it has an internal structure formed by "*number_of_wheels*", "*load_capacity*" and "*trademark*". Definition 2 is applied to the concepts case. The level of abstraction of the concept "*car*" is the vector *[1,D,d]*. *c_abstraction(car)=[1,D,d]* means that two attributes are abstract and one constant. "*car*" is an abstract concept. Suppose we define a new concept "*my_car*" with the attributes: *number_of_wheels*=4, *load_capacity*=1000 Kg and *trademark*="*Ford"*. "*my_car*" is an instance because all attributes are defined and were a defined instance of "*my_car*" is "*Ford*". With the given concept definitions, it is also easy to check that they are well-defined and that the following relations hold: *upper(vehicle, bicycle), upper(vehicle, car), upper(vehicle, truck), lower(bicycle, vehicle), lower(car, vehicle)* and *lower(truck,vehicle)*. Therefore:

upper_context(car,C)=upper_context(bicycle,C)=upper_context(truck,C)={vehicle}

lower_context(vehicle,C)={bicycle,car,truck)

lower_context(bicycle, C)= lower_context(car,C)=upper_context(truck, C)= ϕ.

A taxonomy in our framework is a set of well-defined concepts hierarchically presented with a set of axioms (i.e., laws that always hold between the attributes of the same or different concepts). We now extend our formalisation to model taxonomies:

Definition 8: **Well–connectedness:** Let *C* be a non-empty set of well-defined concepts. These are said to be well-connected, written *well-connected(C)*, if the following conditions hold: (i) $\forall x \in C$, *categorical_context(x,C)* $\neq \phi$; and

(ii) $\exists y \in C$ *s.t. top_level(y,C).*

Definition 9*:* **Upper/lower level set:** Let *C* be a non-empty set of well-defined concepts, and let *S* be a non-empty subset of *C*, $S \subseteq C$. The upper level set of concepts for *S* in *C*, written *upper_set(S,C)* is defined as $\bigcup_{c \in S} \text{upper\_context}(c, C)$. Similarly, the lower level set of concepts for S in C, written *lower_set(S,C),* is defined as $\bigcup_{c \in S} \text{lower\_context}(c, C)$

Definition 10: **Well-defined axiom:** Let *C* be a non-empty set of well-defined concepts, and let *AX* be a rule/law defined over a non-empty set of concepts $S \subseteq C$ through the relationships between the attributes of the elements of *S*. *AX* is said to be well-defined for *S* in *C*, written *well-defined(AX,S,C)* if the following conditions hold:

i) $\forall x \in S$, *abstract(x)*; and

ii) *scope(lower_set(S,C), AX).*

where *scope(y,z)* stands for the logical sentence '*the set of concepts y hold the axiom z'*

Definition 11*:* **Taxonomic ontology structure:** Let *C* be a non-empty set of well-defined and well-connected concepts, and let *AX* be a set of well-defined axioms over a subset $S \subseteq C$. A taxonomic ontology structure, written *TAX*, is defined as the pair <*C, AX*>.

Example 2 illustrating Definitions 8 to 11:

Let *C* be the set *{vehicle, bicycle, car, van, truck, racing_bicycle, mountain_bicycle, touring_bicycle, family_car, sportwagon}*, and let *S* be the set *{mountain_bicycle, car}.*

Then:

categorical_context(vehicle,C)={ bicycle, car, van, truck} $\neq \phi$
categorical_context(bicycle,C)={vehicle, car, van, truck} $\neq \phi$
categorical_context(car,C)={vehicle, bicycle, van, truck} $\neq \phi$
categorical_context(van,C)={vehicle, bicycle, car, truck} $\neq \phi$
categorical_context(truck,C)={vehicle, bicycle, car, van} $\neq \phi$
top_level(vehicle,C) holds.
So, *well-connected(C)* holds.

upper_set(S, C)=upper_context(mountain_bicycle, C) $\cup$ *upper_context(car, C) = { vehicle, bicycle}* *{vehicle} = {vehicle, bicycle}*;

lower_set(S,C)=lower_context(racing bicycle, C) $\cup$ *lower_context(car,C) =* $\cup$ *{family car, sportwagon} = {family car, sportwagon}*.

Let *S'* be *{car}*. Then, the axiom *A="the maximum number of car passengers is 8"* is a well-defined one.

We can also define a taxonomic structure as follows:

The following taxonomy might be defined: *<C,AX>= <{vehicle, bicycle, car, van, truck, racing_bicycle, mountain_bicycle, touring_bicycle, family_car, sportwagon, sport_car}, {"the maximum number of car passengers is 8"}>.*

In the next section, we extend the definitions of this section to produce quality metrics that can then be algorithmically applied. This is also shown in the next section.

4 Modelling Taxonomy Quality and Algorithmic Evaluation Of Taxonomies

We define formal parameters for assessing taxonomic ontologies. We deal with the issue of defining a set of concepts grouped into different categories by attending to their respective degree of abstraction. We make use of the 'closest category' concept following Aristotle-based terminology in order to precisely classify concrete/abstract entities.

4.1 Modelling Quality Features

A taxonomic ontology is well-categorised when attributes of all concepts in the taxonomy, have their range of possible values formed by the union of ranges of possible values in corresponding children concepts. This is formalised in the following:

Definition 12: Well-categorised: A non-empty set *C* of concepts is well-categorised if *C* is a taxonomic ontology structure with the following property:

i) $\forall c \in C, \forall a \in IN(c),$

ii) $POSSIBLE\_VALUES(c,a) = \bigcup_{c' \in children(c)} POSSIBLE\_VALUES(c',a)$

Definition 13: **Closest upper/lower context:** Let *c* and *c'* be well-defined concepts. *c'* is the closest-up-context of *c*, written *Closest_upper(c',c)* if all the following conditions hold:

i) $IN(c) = IN(c')$;

ii) $\forall x \in IN(c),\ a\_abstraction(c,x) \leq a\_abstraction(c',x);$ and

iii) $Cardinal(\{x \in IN(c) \text{ s.t. } a\_abstraction(c,x) < a\_abstraction(c',x)\})=1$.

Similarly, *Closest_lower(c',c)* can be defined as holding the following conditions:

i) $IN(c) = IN(c')$;

ii) $\forall x \in IN(c),\ a\_abstraction(c,x) \geq a\_abstraction(c',x);$ and

iii) $Cardinal(\{x \in IN(c) \text{ s.t. } a\_abstraction(c,x) > a\_abstraction(c',x)\})=1$.

Through definitions 15 to 19, we formalise quality criteria for taxonomic ontologies. One indicator for quality is how 'real' the concepts are. This can be assessed by examining how easy it is to instantiate them. We ue the notion of 'usefulness' to describe how useful some (abstract) categories are for classifying others (i.e., more concrete ones). The presence of both types abstract and instance concepts will be considered. Another indicator of a 'good taxonomic ontology' is the depth of abstraction. The more levels of abstraction (including that of instances) in a taxonomy, the better it is.

Definition 14: **Instantiated/abstract taxonomic ontology structure:** Let *T* be a well-categorised taxonomic ontology structure with *C* being its set of well-defined and well-connected concepts. *T* is said to be instantiated, written *instantiated(T)*, if $\exists x \in C$ *s.t. instance(x).* *T* is said to be abstract, written *abstract_t(T)*, if $\forall x \in C$ *,abstract(x).*

Example: Let *T* be a well-categorised taxonomic ontology structure whose concepts are *{car, sport_car, family_car, sportwagon, my_family_car, my_sportwagon}*. We define "*my_family_car*" as *number_of passengers=6, number_of_wheels=4, load_capacity=1000 Kg and trademark=Ford* and we define *"my_sportwagon" as number_of passenger=4, number_of:wheels=4, load_capacity=2000 Kg and trademark=Peugeot*. *T* is instantiated because *instance(my sportwagon)* and *instance(my family car)* are true.

Definition 15: **Completely/partially instantiated taxonomic ontology structure:** Let *T* be an instantiated taxonomic ontology structure with *C* being its set of well-defined and well-connected concepts. *T* is said to be completely instantiated, written *c_instantiated(T)*, if the following holds:

$\forall x \in C$ *s.t.* $lower\_context(x,C) = \phi \rightarrow instance(x).$

Otherwise, *T* is said to be partially instantiated, namely,

T is partially instantiated, written *p_instantiated(T)*, if

$$\exists x \in C \ \ s.t. \ \ lower\_context(x,c) \neq \varnothing \Rightarrow abstract(x)$$

We can define a measure for the degree of instantiation associated to the taxonomic ontology. We consider the set of instances with respect to the set of abstract concepts situated at the lowest abstraction level:

Definition 16: **Degree of instantiation:** Let *T* be a well-categorised taxonomic ontology structure with *C* being its set of well-defined and well-connected concepts; let *LOWER(C) = {x ∈ C s.t. lower_context (x,C) = ϕ}* and let *INSTANCES(C) = {x ∈ C s.t. instance(x)}.*

Then, the degree of instantiation of *T*, written *degree_inst(T),* is defined as *Cardinal(INSTANCES(C))/ Cardinal(LOWER(C)).*

Example: Suppose that *T={bicycle, racing_bicycle, mountain_bicycle}.* "*mountain_bicycle*" and "*racing_bicycle*" are instances, whereas *lower(mountain_bicycle)=lower(racing_bicycle)= ϕ.* So, *degree_inst(T)=2/3=0.66*

Another parameter we use to measure the quality of a taxonomic ontology is the completeness. This checks if the sub-categories cover the entire domain to be categorised. A concept *c* is complete if for every attribute there are two or more child concepts. A category is complete if all its concepts are complete.

Definition 17: Well-built taxonomic structure: Let *T* be a taxonomic ontology structure with *C* being its set of well-defined and well-connected concepts. *T* is a well-built taxonomic structure, written *well-built(T),* if

children(c) as the set of all *c'* such that *child (c', c),* where *child(c, c')* holds if i) *upper(c, c')* ii) *not (∃x ∈ C s.t. upper(x, c') Λ upper(x, c))*

Example: Let *C= {vehicle, bicycle, car, sport_car, family_car, racing_bicycle, mountain_bicycle},* then *C* is well-built because:

ABSTRACT(C)= {x ∈ C s.t. abstract(x)}={vehicle, bicycle, car}

Children(vehicle)={bicycle, car, sport_car, family_car, racing_bicycle, mountain_bicycle} then *Cardinal(children(vehicle))=6*

Children(car)={sport_car, family_car} then Cardinal(children(car))=2

Children(bicycle)={racing bicycle, mountain_bicycle} then Cardinal(children(bicycle))=2

Then *C* is a well-built *TAX*.

Furthermore, we can define a measure similar to usefulness for measuring the degree of completeness based on the concepts a taxonomic ontology contains.

Definition 18: Degree of completeness: Let *T* be a taxonomic ontology structure with *C* being its set of well-defined and well-connected concepts. Let *ABSTRACT(C)={x ∈ C s.t. abstract(x)}* and *CHILDREN(C)={x ∈ C s.t. children(x}}.* The degree of completeness of *T*, written *degree_completeness(T),* is defined as *Cardinal(ABSTRACT (C))/ Cardinal(CHILDREN(C)).*

Example: *ABSTRACT(C)={x ∈ C s.t. abstract(x)}={vehicle, bicycle, car}*

CHILDREN(C))={bicycle, car, sport_car, family_car, racing_bicycle, mountain_bicycle} then degree_completeness(T)=Cardinal(ABSTRACT (C))/ Cardinal(CHILDREN(C))=3/6=1/2.

Building and evaluating our taxonomy algorithmically is presented in this section. Building the ontology is based on two concept refinement processes, a bottom-up process generating an upper category from a given concept and a top-down process generating lower

categories from a given concept. The automatic taxonomic evaluation and refinement of the taxonomic structure is based on identifying a well defined set of concepts by analysing the similarities of all involved attributes. This structure then evolves by using the bottom up and top down refinement operations that may be invoked by humans or software agents.

4.2 Bottom Up Taxonomy Construction

The algorithm proposed to form an upper category from a given concept *c* (i.e., to go up in the taxonomic ontology) is the following:

For every well-defined concept, *c*, selected by the user:

1. Generate closest_Upper (*CURRENT_PARENTS, c)* /* per definition 13 */
2. Generate all possible parents in the set *TOTAL_UP*). Each element in the set has attribute from *c* 'undefined' but it includes the range of values to satisfy it (that is, the property of being *closest_up*).
3. If *TOTAL_UP* = Ø check each new concept to be formed, ***c'***, $c \in$ *closest_down*(***c'***). As follows:

   L= *closest_upper_categories=TOTAL_UP\CURRENT_PARENTS*

   For this operation: the objective is to compare ranges of attributes (since the name of the concept is irrelevant at this stage and the attributes names are the same). The range 'undefined' (in *total_up*) will be taken as having the same value as the corresponding attribute of *current_parents*.

   IF L = Ø propose to the user these two options:

   - Modify the range of one of the current parents and revise name if required (***c'*** holding the property of *closest-up* of *c*).
   - **OR:** select *c* as one of the *closest_up* and break out of the 'for' loop

   IF L ≠ Ø

   - If the user selects some concept of L as the *closest_up*, name it and break
   - Else let the user build ***c'*** ensuring that $c \in$ *closest_down*(***c'***)

That is *TOTAL_UP* is the following procedure:

Input: a concept *c*

Output: a set of concepts *P*

P = getParents(c)
While (exists $x \in P$ such that *getParents(x)*) do
$P = P \cup$ *getParents*(*x*)

4.3 Top Down Taxonomy Construction

The algorithm proposed to form a lower category from a given concept *c* (i.e., to go down in the taxonomic ontology) is the following:

For every user-selected well-defined concept, c:

1. Generate closest_lower (*CURRENT_CHILDREN*, c) /* per definition 13 */
2. Generate all possible children in the set *TOTAL_DOWN*. Each child as *an attribute from c* 'undefined' but it includes the range of values to satisfy it (that is, the property of being *closest_down*).

   That is, *TOTAL_DOWN* comes from the following procedure:

   Input: a concept *c*

   Output: a set of concepts *P*

   P = getChildren(c)

   While (exists $x \in P$ such that *getChildren(x)*) do
   $P = P \cup$ *getChildren(x)*

4.4 Fully automatic taxonomic ontology building

Our algorithm proposed to build the ontology is the following:

1. Get the first well-defined concept (*c*) from the user through procedure *acquireConcept* (see below)
2. Set of current concepts (*C*) = *c*
3. While user enters concepts do
   a) ic = acquireConcept
   b) Compare attributes and possible values of the incoming concept (*ic*) with each of *C*, so that the exact position will depend on set of attributes and degree of abstraction of *ic* with respect to those of each element of *C*.

Where *acquireConcept* is the following procedure:

Procedure *acquireConcept*

```
acquire concept-name;
acquire attribute-list;
For each attribute
    acquireAttribute
    check validAttribute;
```

And *acquireAttribute* and *validAttribute* are the following:

Procedure *acquireAttribute*

```
acquire attribute-name;
acquire attribute-possible-values;
```

Procedure *validAttribute*

```
if attribute-name is equal to concept-name, NOT VALID;
if set of attribute-possible-values is empty, NOT VALID;
if attribute is in attribute-possible-values set, NOT VALID;
```

if concept is in attribute-possible-values set, NOT VALID;
else is VALID

5 METHODOLOGY APPLICATION IN THE ROAD SAFETY DOMAIN

We illustrate our enhancement methodology with an example. Three concepts and their corresponding attributes (shown in italics) are used:

Vehicle: *Number_of_wheels: N ∪{0}*

Car: *Number_of_wheels: 4*; *Load_capacity: 1000 Kg*; *Trademark: Any string*; *Number_of_passengers: {1,2,3,4}*.

Sport_car: *Number_of_wheels: 4*; *Load_capacity: 1000 Kg*; *Trademark: Any string*; *Number_of_passengers: {1,2}*; *Acceleration (m/s^2): less than one (≤1)*.

The above concepts are our input and are supposed to belong to the same taxonomical structure so that the links between the closest upper and lower concepts can be discovered. The first step to find potential upper (respectively lower) relations between the new concepts and any existing concepts in the taxonomy, which is initially emtpy. The first processed concept, *"vehicle",* is added to the ontology without any taxonomic link. The next concept, "*car*", has to be compared with "*vehicle*". Since *upper(vehicle, car)* holds (as opposed to *upper (car, vehicle)*), we discover that "car is a type of vehicle".

The next concept, "*sport_car*", is compared to the already included concepts, "*vehicle*" and "*car*". It is easily checked that both *upper (vehicle, sport_car)* and *upper (car, sport_car)* hold so that "*sport_car*" is a type of "*vehicle*" and "*sport_car*" is also type of "*car*". We could either create these two taxonomic links, or calculate which one is the closest context to the concept "*sport_car*". If we try to calculate this, the result will be negative because "*sport_car*" and "*car*", and "*sport_car*" and "*vehicle*" do not have the same attributes. More precisely, *IN(sport_car) ≠ IN* (car) and *IN(sport_car) ≠ IN(vehicle).* Considering that "*is_a*" is a transitive relation and in the taxonomy it is already stated that "car is a type of vehicle", only the link "sport_car is a type of car" will have to be made explicit.

Let's suppose now a new concept is input by the user:

Bicycle: *Number_of_wheels: 2*; *Load_capacity: 200 Kg*; *Trademark: Any string*; *Number_of_passengers: {1,2}*; *Type of road: any path*.

Comparing *Bicycle* to the *Sport_car*, the following links are found: *upper(vehicle, bicycle), equal(bicycle, car).* If we apply the top down algorithm with "*bicycle*" as input, to extend the ontology, a new concept called "*tandem*" will be obtained. This has initially the same attributes as "*bicycle*" with of values of its attribute *number_of_passengers* restricted to one single value (2):

Tandem: *Number_of_wheels: 2*; *Load_capacity: 200 Kg*; *Trademark: Any string*; *Number_of_passengers: 2*; *Type of road: any path*.

Thus, *upper(bicycle, tandem)* and *upper (vehicle, tandem)* hold. It is also checked that "*bicycle*" is an upper concept for "*tandem*". The final taxonomic structure of the ontology is shown in Figure 1.

The OWL implementation of the vehicle ontology can be found in the Annex section. Another usage of our methodology is to fix and rebuild a malformed taxonomy. This usage is illustrated in the rest of this section.

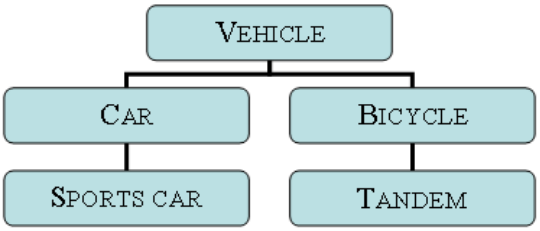

Figure 1. The final taxonomic structure for the running example

We use as an example the taxonomy shown in Figure 2 and assume that its concepts have the following attributes:
Road_sign: *Shape: unknown*; *Colours: unknown*; *Message: unknown*.
Prohibition: *Shape: round*; *Colours: red, white and black*; *Message: unknown*.
Maximum_120Km/h: *Shape: round*; *Colours: red, white and black*; *Message: "spped_limit_of_120Km/h"*; *Maximal_speed: 120Km/h*.
Speed_limit: *Shape: round*; *Colours: red, white and black*; *Message: unknown*; *Maximal_speed: unknown*.
Obligation: *Shape: unknown*; *Colours: blue and white*; *Message: unknown*.
Obligatory_direction: *Shape: round*; *Colours: blue and white*; *Message: unknown*; *Direction: unknown*.

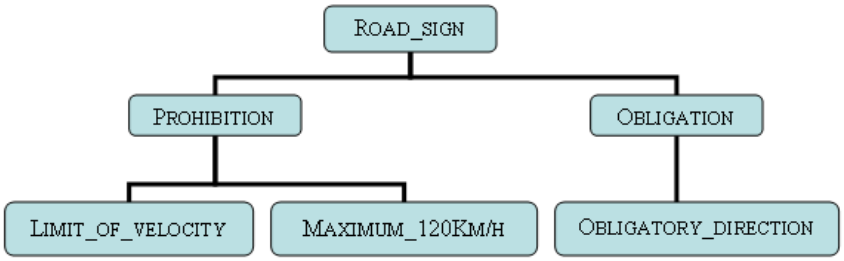

Figure 2. A road sign taxonomy

The top-down algorithm is first applied. With "*road_sign*" as the root concept of our taxonomy, we search now for all children concepts. Given that *upper(prohibition, obligation)* and *upper(obligation, prohibition)* return a false value and both have the same parent, we conclude *equal(prohibition, obligation)* and the corresponding taxonomic links are generated. Now, the concept "*prohibition*" has two children: "*speed_limit*" and "*maximum_120Km/h*". However, *upper(speed_limit, maximum_120Km/h)* is true, because (1) they have the same attributes (2) two of them have the same values and (3) the values for other two attributes of the concept "*maximum_120Km/h*" are included in the range of values defined for the respective attributes for "*speed_limit*" as in "*maximum_120Km/h*". So, *closest_upper(speed_limit, maximum_120Km/h)* holds. Then, the taxonomy is reordered by setting "*maximum_of_120Km/h*" as a child of "*speed_limit*". The concept "*obligation*" has only one child, "*obligatory_direction*". The predicate *upper(obligation,*

obligatory_direction) holds, and no changes are required. The taxonomy of Figure 2 is enhanced to produce the well-defined taxonomy shown in Figure 3.

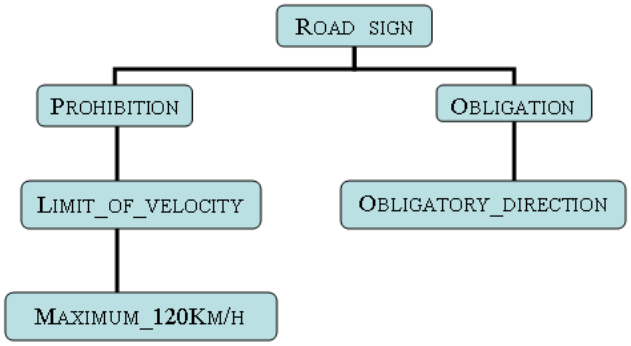

Figure 3. The corrected road sign taxonomy

5.3. Deployment Guidelines of the evaluation framework

The taxonomy development support framework we provide can be used as a complementary tool to support and monitor an incremental taxonomy development process. It can be used to ensure that taxonomies are delivered and ready to use, without the need for a separate testing phase. Operating during the development of the taxonomy, the framework has the added advantage that it makes use of domain experts when they are already available. It therefore does not incur any additional cost in hiring expertise or providing new testing data later. To illustrate, how it can be applied to a practical setting, let us consider the following: Let there be given a sequence of *n* instances (I's) representing the input to the expert in charge of developing a taxonomy. Assume that they were presented in the given order during the knowledge acquisition process: $I_1\ I_2\ I_3\ \ldots\ \boldsymbol{I_{M1}}\ \ldots\ \boldsymbol{I_{M2}}\ \ldots\ \boldsymbol{I_{M3}}\ \ldots\ \boldsymbol{I_{M4}}\ \ldots\ \boldsymbol{I_{M5}}\ \ldots\ \ldots\ \boldsymbol{I_{Ms}}\ \ldots\ I_n$ Where the instances in bold face were misclassified by the partial taxonomy at the time of first presentation and a new concept to correctly classify the respective case was added. Viewed across the entire development of the taxonomy, we see *s* misclassified cases which resulted in the addition of a maximum number of concepts, *s*. E.g., assume that $\boldsymbol{I_{M3}}$ resulted in a new concept being added which is the parent concept being added after $\boldsymbol{I_{Ms}}$ was incurred. In that situation, we can use the entire sequence of cases seen between $\boldsymbol{I_{M3}}$ and I_n. as input to the amendment support algorithm. To deploy a monitoring module that operationalises algorithms 1 and 2, the stream of cases has to be input to the module with the expert's amendments to the taxonomy. In particular, the monitor module requires information on how the data stream is interleaved with the taxonomy development. Special cases that trigger amendment to the taxonomy need to be time stamped to enable the sequence regeneration and analysis.

6 DISCUSSION AND FUTURE WORK

We present in this paper a methodology to construct ontologies as well as a formal approach where the user can define the alphabet where (s)he can form the terms from. The methodology is flexible. It starts from a set of concepts, which are defined through their

corresponding valued attributes. Similarities between concepts are measured through the similarity amongst their attributes and their corresponding values to automatically build a taxonomy ontology. A significant feature of our methodology is that it allows for refining or evolving the taxonomy by making use of both top-down and bottom-up approaches to concept discovery during the construction process. This work is of particular significance to incremental approaches to knowledge models development e.g. in medical systems (Compton, Peters et al. 2006; Bichindaritz and Montani 2009), in call management (Wobcke, Chan et al. 2006) and mechanical engineering (Beydoun, Hoffmann et al. 2010). The concomitant modification of the taxonomy is carried out by following a set of taxonomic principles. In the bottom-up approach, a "closest up" concept is produced. In the top-down approach, a "closest down" concept is generated. We adopt both top-down and bottom up concurrently, accommodating human tendencies in expressing domain knowledge. The full potential of the methodology has been illustrated through an example in the traffic domain (i.e., road safety). It has been shown that the methodology allows for building taxonomies in an automatic way and to refine them either manually or automatically. It is clear to us that this methodology can also be used to improve the quality of existing ontologies. The only requirement is that they are written in OWL, which is a reasonable one, as this ontology language is the standard de facto for ontology development (Grau et al. 2008). Despite the significance of taxonomies in many software applications, to support the development of a wider range of knowledge bases and information systems, richer abstractions are still needed. Therefore, in our future work we will extend the algorithm presented to include other types of semantic relations such as mereology or topology as further work. Another extension we are considering is to combine the approach here presented, with the evaluation of integrated ontologies from multiple sources as we had illustrated earlier in (Beydoun et al. 2006; Beydoun 2009).

Acknowledgments

This work was supported in part by the Spanish Government (under projects TIN2006-14780 and PT-2006-055-24ICPP and the Region of Murcia under project BIO-TEC 06/01-005) and Australian Research Council (Grant DP0878172)

REFERENCES

Balke, W. T., Wagner, M. (2004). "Through different eyes: assessing multiple conceptual views for querying web services". *WWW '04*: Proceedings of the 13th international World Wide Web, New York, NY, USA, ACM: 196–205

Beydoun, G., Gonzalez-Perez, C., et al. (2006). "Developing and Evaluating a Generic Metamodel for MAS Work Products". Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications. A. Garcia, R. Choren, C. Lucenaet al. Berlin, Springer-Verlag. LNCS 3914: 126-142.

Beydoun, G. (2009). "Formal concept analysis for an e-learning semantic web". Expert Systems with Applications 36(8).

Beydoun, G., Hoffmann, A., et al. (2010). "Automating dimensional tolerancing using Ripple Down Rules (RDR)." Expert Systems with Applications 37(7): 5101-5109.

Beydoun, G., Lopez-Lorca, A. et al. (2011). "How do we measure and improve the quality of a hierarchical ontology?" Journal of Systems and Software 84 (12): 2363-2373.

Borst, W. (1997). "Construction of Engineering Ontologies". *PhD thesis*, University of Twente, Enschede.

Brewster, C., K. O'Hara, et al. (2004). "Knowledge Representation with Ontologies: The Present and Future." IEEE Intelligent Systems 19(1): 72-81.

Chamiel, G., Pagnucco, M. (2008). "Exploiting ontological information for reasoning with preferences" *Proceedings of the Fourth Multidisciplinary Workshop on Advances in Preference Handling*.

Fortuna, B., Grobelnik, M., Mladenic, D. (2007). "*OntoGen: Semi-automatic Ontology Editor"*. In Human Interface and the Management of Information. Interacting in Information Environments. p. 309-318. Springer Berlin / Heidelberg.

Grau, B., Horrocks, I., et al. (2008). "OWL 2: The next step for OWL, Web Semantics: Science, Services and Agents on the World Wide Web", Semantic Web Challenge 2006/2007, 6(4): 309-322.

Grogono, P., Batarekh, A., et al. (1991). "Expert system evaluation techniques: a selected bibliography." Expert Systems 8(4): 227-239.

Guarino, N., Welty, C. (2001). "Supporting Ontological Analysis of Taxonomic Relationships," *Data and Knowledge Engineering*, 39 (1): 51-74.

Guarino, N., Welty C. (2004): "An Overview of OntoClean". *In S. Staab & R. Studer (Eds.),* The Handbook on Ontologies (pp. 151-172). Berlin: Springer.

Menzies, T. (1998). Evaluation Issues With Critical Success Metrics. 11th Banff Knowledge Acquisition for Knowledge Base System Workshop (KAW99), Canada, SRDG Publications.

Middleton, S., Shadbolt, N., De Roure, D. (2004). "Ontological user profiling in recommender systems". *ACM* Trans. Inf. Syst., 22(1): 54–88

Nick, M., Althoff, K., et al. (1999). Facilitating the Practical Evaluation of Organizational Memories Using the Goal-Question-Metric Technique. 12th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW99), Canada, SRDG publications.

Ning, H., Shihan, D. (2006). Structure-Based Ontology Evaluation. IEEE International Conference on e-Business Engineering (ICEBE'06): 132-137.

Punera, K., Rajan, S., et al. (2006). Automatic Construction of N-ary Tree Based Taxonomies. Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06), Austin, University of Texas.

Sadraei, E., Aurum, A., et al. (2007). "A field study of the requirements engineering practice in Australian software industry" *Requirements Engineering* 12(3):145-162.

Schickel-Zuber, V., Faltings, B. (2007). "Using hierarchical clustering for learning the ontologies used in recommendation systems". *KDD 2007*, 599–608

Staab, S., Schnurr, H. et al. (2001). "Knowledge Processes and Ontologies," *IEEE Intelligent Systems*, 16 (1).

Völker, J., Vrandecic D., Sure, Y. (2005). "Automatic Evaluation of Ontologies (AEON)". *Y. Gil et al. (Eds.): Proceedings of the 4th International Semantic Web Conference (ISWC 2005), LNCS 3729: 716-731*, Springer Verlag Berlin-Heidelberg.

APPENDIX A

a) OWL implementation of the "vehicle" ontology

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/Ontology1221644678.owl#"
  xml:base="http://www.owl-ontologies.com/Ontology1221644678.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Vehicle"/>
  <owl:Class rdf:ID="Sport_car">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="acceleration"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Car"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >2</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="number_of_passengers"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Tandem">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Bicycle"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >2</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#number_of_passengers"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Bicycle">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >200</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="load_capacity"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Vehicle"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="number_of_wheel"/>
        </owl:onProperty>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >2</owl:hasValue>
```

```
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >2</owl:maxCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#number_of_passengers"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Car">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#load_capacity"/>
        </owl:onProperty>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1000</owl:hasValue>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Vehicle"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >4</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#number_of_wheel"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >4</owl:maxCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#number_of_passengers"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:DatatypeProperty rdf:about="#acceleration">
    <rdfs:domain rdf:resource="#Sport_car"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#number_of_wheel">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#Vehicle"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#load_capacity">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Car"/>
          <owl:Class rdf:about="#Bicycle"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="trademark">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Bicycle"/>
          <owl:Class rdf:about="#Car"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
```

```
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#number_of_passengers">
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Bicycle"/>
          <owl:Class rdf:about="#Car"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range>
      <owl:DataRange>
        <owl:oneOf rdf:parseType="Resource">
          <rdf:rest rdf:parseType="Resource">
            <rdf:rest rdf:parseType="Resource">
              <rdf:rest rdf:parseType="Resource">
                <rdf:rest rdf:parseType="Resource">
                  <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
                  >5</rdf:first>
                  <rdf:rest rdf:parseType="Resource">
                    <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
                    >6</rdf:first>
                    <rdf:rest rdf:parseType="Resource">
                      <rdf:rest rdf:parseType="Resource">
                        <rdf:rest     rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#nil"/>
                        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
                        >8</rdf:first>
                      </rdf:rest>
                      <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
                      >7</rdf:first>
                    </rdf:rest>
                  </rdf:rest>
                </rdf:rest>
                <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
                >4</rdf:first>
              </rdf:rest>
              <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
              >3</rdf:first>
            </rdf:rest>
            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
            >2</rdf:first>
          </rdf:rest>
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >1</rdf:first>
        </owl:oneOf>
      </owl:DataRange>
    </rdfs:range>
  </owl:DatatypeProperty>

</rdf:RDF>
```

b) A road sign taxonomy

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.owl-ontologies.com/Ontology1222678200.owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.owl-ontologies.com/Ontology1222678200.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Prohibition">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >red, white and black</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Colours"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >round</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Shape"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Road_sign"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Maximum_120km-h">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Maximal_speed"/>
        </owl:onProperty>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >120</owl:hasValue>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >limit_of_velocity_of_120Km/h</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Message"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Prohibition"/>
  </owl:Class>
  <owl:Class rdf:ID="Limit_of_velocity">
    <rdfs:subClassOf rdf:resource="#Prohibition"/>
  </owl:Class>
  <owl:Class rdf:ID="Obligatory_direction">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >round</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#Shape"/>
        </owl:onProperty>
      </owl:Restriction>
```

```
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Obligation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Obligation">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >blue and white</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#Colours"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Road_sign"/>
  </owl:Class>
  <owl:DatatypeProperty rdf:about="#Maximal_speed">
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Maximum_120km-h"/>
          <owl:Class rdf:about="#Limit_of_velocity"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#Message">
    <rdfs:domain rdf:resource="#Road_sign"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#Shape">
    <rdfs:domain rdf:resource="#Road_sign"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Direction">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Obligatory_direction"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#Colours">
    <rdfs:domain rdf:resource="#Road_sign"/>
  </owl:DatatypeProperty>

</rdf:RDF>
```

c) The corrected road sign taxonomy

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.owl-ontologies.com/Ontology1222678200.owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.owl-ontologies.com/Ontology1222678200.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Prohibition">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >red, white and black</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Colours"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >round</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Shape"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Road_sign"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Maximum_120km-h">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Limit_of_velocity"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Maximal_speed"/>
        </owl:onProperty>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >120</owl:hasValue>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >limit_of_velocity_of_120Km/h</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Message"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Limit_of_velocity">
    <rdfs:subClassOf rdf:resource="#Prohibition"/>
  </owl:Class>
  <owl:Class rdf:ID="Obligatory_direction">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >round</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#Shape"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
```

```
      <owl:Class rdf:ID="Obligation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Obligation">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >blue and white</owl:hasValue>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#Colours"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Road_sign"/>
  </owl:Class>
  <owl:DatatypeProperty rdf:about="#Maximal_speed">
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Maximum_120km-h"/>
            <owl:Class rdf:about="#Limit_of_velocity"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#Message">
    <rdfs:domain rdf:resource="#Road_sign"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#Shape">
    <rdfs:domain rdf:resource="#Road_sign"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Direction">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Obligatory_direction"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#Colours">
    <rdfs:domain rdf:resource="#Road_sign"/>
  </owl:DatatypeProperty>
</rdf:RDF>
```