Deliverable number:	D4.4
Deliverable title:	Taxonomy-based collaborative filtering algorithms

Delivery date:	2011/07/31 (Extension date: 2011/10/31)
Submission date:	2011/11/02
Leading beneficiary:	Universidad Politécnica de Madrid
Status:	Version 1.2 (final)
Dissemination level:	PU (public)
Authors:	Jorge Gracia, Esther Lozano, Jochem Liem, Diego Collarana, Oscar Corcho, Asunción Gómez-Pérez, and Boris Villazón

Project number:	231526
Project acronym:	DynaLearn
Project title:	DynaLearn - Engaging and informed tools for learning conceptual system knowledge
Starting date:	February 1st, 2009
Duration:	36 Months
Call identifier:	FP7-ICT-2007-3
Funding scheme:	Collaborative project (STREP)

Abstract

In DynaLearn, semantics of the QR models ingredients is made explicit by representing them as terms in ontologies. That easies the task of exploring the knowledge contained in the models, enabling rich comparisons among them. The facts that the user explicitly represents in the model constitutes the asserted ontology. Nevertheless, logical rules can be applied to these facts in order to extract other knowledge (inferred facts) that was not made explicit by the modeller. Taxonomical reasoning techniques can make emerge these inferred facts. In the Semantic Technologies module in DynaLearn, the exploration of taxonomic structures and the application of taxonomical reasoning techniques play a major role.

This document describes the task of integrating taxonomic reasoning in DynaLearn in order to enrich the results presented to users by discovering additional semantic information that is not explicit in the QR models initially. This enables: (1) a better identification of similar terms between models, which directly benefits feedback and collaborative filtering, (2) detection of inconsistencies between models, which enriches the information given to the user during semantic feedback, and (3) classification of instances during the grounding process. All these aspects are analysed in the document.

As an addition, and to complete the cycle of WP4 deliverables, we annex in this document a description of the test plan that we created and applied regularly to the different components of the ST module.

Internal review

- René Bühling, Multimedia Concepts and Applications, University of Augsburg (UAU), Germany.

- Petya Borisova, Institute of Biodiversity and Ecosystem Research at the Bulgarian Academy of Sciences (IBER)

Acknowledgements

We thank our internal reviewers for their careful review and useful comments.

Document History

Version	Modification(s)	Date	Author(s)
0.1	Initial draft	2011-07-07	Jorge Gracia
0.2	Instance grounding & deriving domain concepts	2011-09-26	Jochem Liem
0.3	Reasoning during semantic feedback (draft)	2011-10-13	Esther Lozano
0.4	Taxonomic evaluation	2011-10-17	Esther Lozano
0.5	Inference during ontology matching	2011-10-18	Jorge Gracia
0.6	Taxonomic evaluation revised	2011-10-19	Jorge Gracia
0.7	Sections 1, 2, and 4 revised	2011-10-20	Jochem Liem
0.8	Annex A added	2011-10-20	Diego Collarana
0.9	Version for internal review	2011-10-20	Jorge Gracia
1.0	Section 3 revised	2011-10-21	Jochem Liem
1.1	Annex A revised	2011-10-31	Diego Collarana
1.2	Version corrected after internal review (final)	2011-10-31	Jorge Gracia

Contents

1. Introduction

According to the Description of Work (DoW), Task 4.4 is aimed to “integrate taxonomic reasoning in DynaLearn” in order to “improve the accuracy of the results presented to users, since they will allow relating topics that would not be considered as similar initially due to the absence of taxonomic models in the algorithms”.

The Semantic Technologies (ST) module in DynaLearn enables the online storage of user QR models, for their later reuse as well as to serve as basis for semantic feedback generation [10,6]. In fact, at modelling time, comparisons with other QR models in the repository can be established in order to detect expert knowledge that can enrich or correct the current model. This is, precisely, the task of the semantic feedback component [10]. Furthermore, semantic feedback needs a previous task for selecting relevant expert knowledge from which the feedback will be extracted. This is the recommendation process [9].

For the recommendation task in DynaLearn, Collaborative Filtering techniques are used to filter the information that is given back to the user when he asks for semantic-based feedback, primarily by selecting suitable reference models as source of that information. In general, Collaborative Filtering aims to provide personalized recommendations to users based on information obtained from similar like-minded users [1]. Such recommendation algorithms, largely discussed in [9], rely upon comparisons which are established between the user model and the knowledge contained in the semantic repository, in order to compute similarities that are used to select suitable reference models for generating suggestions.

In the process of identifying relevant knowledge for feedback, the exploration of taxonomic structures and the application of semantic reasoning techniques play a central role. Precisely , the objective of this deliverable is to describe the application of taxonomic reasoning[1] in different parts of the grounding, recommendation, and feedback process. As it is described in [9], our algorithms are not limited to user-based ratings, tags and keywords as other recommendation algorithms do, but also consider the knowledge characteristics of models created by users. This knowledge is described in terms of ontologies. Ontologies can express facts as logical statements and inference rules can be applied to enrich the asserted knowledge with new inferred one not explicitly declared in the user models. A very simple example is offered by transitive entailment: A subclassOf B and B subclassOf C $\Rightarrow$ A subclassOf C

Despite the fact that the title of the document refers to collaborative filtering techniques, the application of semantic reasoning in DynaLearn has a larger scope and touches several components of the ST module:

- **Ontology mapping** techniques. Ontology mapping techniques (see [6,10]) bridge across different knowledge models to enable the obtaining of relevant semantic feedback.

- **Semantic-based feedback**. During the generation of semantic feedback, we can use the reference model to detect taxonomic errors in the learner model by applying semantic reasoning techniques.

- **Grounding**. For Learning Spaces 1 to 5, hierarchy of the entities is not explicit and supertypes are hidden, which hampers the semantic reconciliation between models required to enable recommendations and feedback. There are certain cases in which the mechanisms we used

[1] We use *taxonomic reasoning* and *semantic reasoning* in an interchangeable way along the document.

for multiword grounding [6] can be used to infer common types of independent instances, which lead to a better modelling scheme.

The application of lightweight semantic inference in the above scenarios results is a richer recommendation and feedback, as much extra knowledge relevant to the user is discovered during the process.

The rest of this document is organised as follows. In Section 2, inference during the ontology matching process is presented. Section 3 focuses on the role of taxonomic reasoning during semantic feedback. Section 4 describes our algorithms to classify instances during grounding. Finally, Sections 5 and 6 close the document with the conclusions and discussion of results respectively.

2. Inference during ontology matching

As it was described in Deliverable D4.1, the Semantic Techniques (ST) module in DynaLearn requires the intervention of a tool for ontology alignment. In fact, we need techniques to reconcile different QR models in order to analyse the similarities and differences between them and, based on that, provide useful feedback during the modelling construction. To that end, the use of well-established ontology matching techniques was analysed in Deliverable D4.1. As result of this study we opted to use CIDER CIDER (Context and Inference baseD ontology alignER) [4] in DynaLearn. In the rest of this section we briefly introduce CIDER, we comment on the use of semantic reasoning during the matching process in CIDER, and finally explore the effect of semantic reasoning in the feedback by exploring some real examples.

2.1. CIDER system

CIDER [4] is designed to discover equivalence relationships between ontology terms (ingredients of qualitative models in the case of DynaLearn). The system performs ontology matching between two given ontologies, producing a set of correspondences as output. Figure 1 shows a schematic view of the approach. O1 and O2 represent the input ontologies. M is the matrix of resultant comparisons among ontology terms, and A is the extracted alignment.

The first step is to extract the ontological context of each involved term, that is, their synonyms, textual descriptions, related terms in the taxonomy, etc. This process is enriched by applying lightweight inference mechanism, in order to add more semantic information that is not explicit in the asserted ontologies, as we will see later in Section 2.2. By lightweight inference we mean that the number of applied inference rules will not be high, because high inference levels are very time consuming, thus hampering scalability. One can found in *transitive inference* level the best balance between quality of results and time response in CIDER, therefore it is adopted as default mode [4]. Also CIDER behaves reasonably well at level of RDFS inference rules. However, in scenarios where time response is not an issue, or ontologies are small, higher inference levels can be applied.

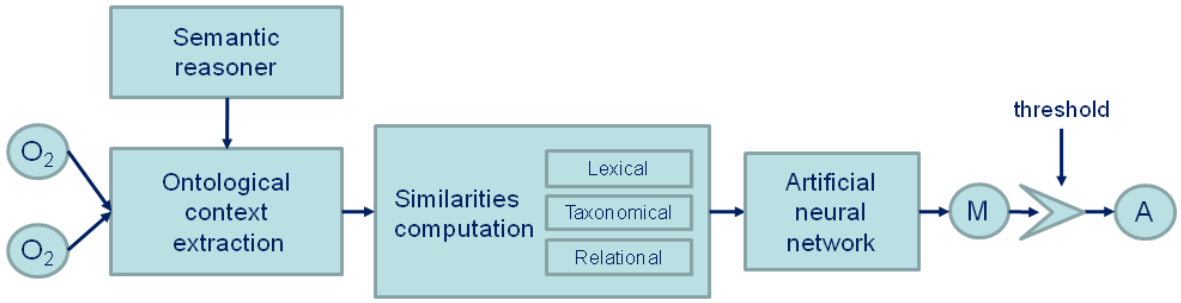

Figure 1: Scheme of CIDER

The next step, once the ontological context has been extracted, is to compute a semantic similarity measure between each pair of aligned terms. To that end, several elementary similarity measures are computed first. They involve lexical similarity between labels, as well as structural similarities, based on vector space modelling techniques [13], between the taxonomies and the relations among terms.

In order to avoid the necessity of manual tuning when combining these measures, an artificial neural network (ANN) was trained to that end. ANNs constitute an adaptive type of systems composed of interconnected artificial neurons which change the structure based on external or internal information that flows through the network during a learning phase [12]. In CIDER, the above mentioned similarities are the input to the neural network (a multilayer perceptron) that combines them into a final similarity degree for each pair of terms. These similarities are organised in a matrix (M, in Figure 1) and, finally, an alignment (A) is extracted from it, filtering out the results below a certain threshold.

In principle, the ANN in CIDER has been trained with a subset of the reference alignments provided in the benchmark track of the Ontology Alignment Evaluation Initiative[2] (OAEI). Nevertheless, it can be trained with data from any other domain, so the system can be adapted in the future to the environmental (or other) domain.

CIDER has been developed in Java (which favours its integration in DynaLearn ST component), extending the Alignment API [2]. The input are ontologies expressed in OWL (Web Ontology Language), and the output is served as a file expressed in the alignment format [2], although it can be easily translated to other formats as well.

2.2. Inference to enhance similarity computation

The *ontological context extraction* phase in the process described in Section 2.1 is aimed to retrieve as much different semantic information about the ontology terms as possible, to allow a precise comparison later. In principle, the nearest neighbouring ontological elements are extracted, however enriched by adding the inferred facts deduced from a reasoning service (as we will discuss later in this section).

The extracted ontological elements comprise: URI, identifying label, synonym labels, and glosses (descriptions in natural language). Additionally, other elements that depend on the type of ontology term are extracted:

- For classes: Hypernyms and hyponyms (the direct ones and the others inferred from the hierarchy), properties for which the term is the domain, and classes related to the explored term by any other property.
- For properties: Superproperties and subproperties of the hierarchy, as well as domains and ranges of the explored term.
- For individuals: Associated concepts, properties, and property values.

The way we access and retrieve the ontological information of the ontology term depends on the nature of the accessed source of knowledge. When used in the context of DynaLearn, the ontologies

[2] http://oaei.ontologymatching.org/2011/

are loaded as model objects in the Jena semantic framework[3], to be interrogated later by using SPARQL [11] and Jena Ontology API.

As mentioned before, we apply an external reasoning service during this extraction step. To that end, Jena built-in reasoner[4] is used, although other reasoners can be used instead. It allows us to load and interrogate the inferred ontology instead of the asserted one. Different levels of inference can be applied:

- No inference. Only elements from the asserted ontology are extracted.
- *Transitive inference*. All super/subterms reached by transitivity are added to the extracted ontological context.
- *RDFS inference*. Apart from transitivity, other RDFS entailments are applied [7] (e.g., given *p* a property, *c* a class and *a* and *b* instances, *c* is domain of *p* and *a* related to *b* by $p \Rightarrow a$ is an instance of *c*).
- *OWL inference*. In addition to transitive and RDFS entailments, other OWL rules are considered (e.g., given the classes *c,d,e*, *c* = unionOf(*d,e*) $\Rightarrow$ *d* and *e* are subclasses of *c*).

Other levels are possible as, for example, the different subsets of OWL rules allowed by Jena built-in reasoner. CIDER applies lightweight inference because it uses low inference levels (transitive, or RDFS) preferentially to operate.

Let us see a simple example to illustrate the benefits of enriching the ontological context with inferred facts before the similarity computation. Suppose that we have two models (learner and expert) in which the animal *lion* is semantically represented in two different ways (see Figure 2). Suppose that we want to discover semantic equivalences between the two models (for instance, to link the learner model with the expert one in order to derive semantic feedback, see [10]). Then our similarity computation should be able to discover that “lion” in model 1 (learner) corresponds to “Panthera leo” (the scientific name for lion) in model 2 (expert).

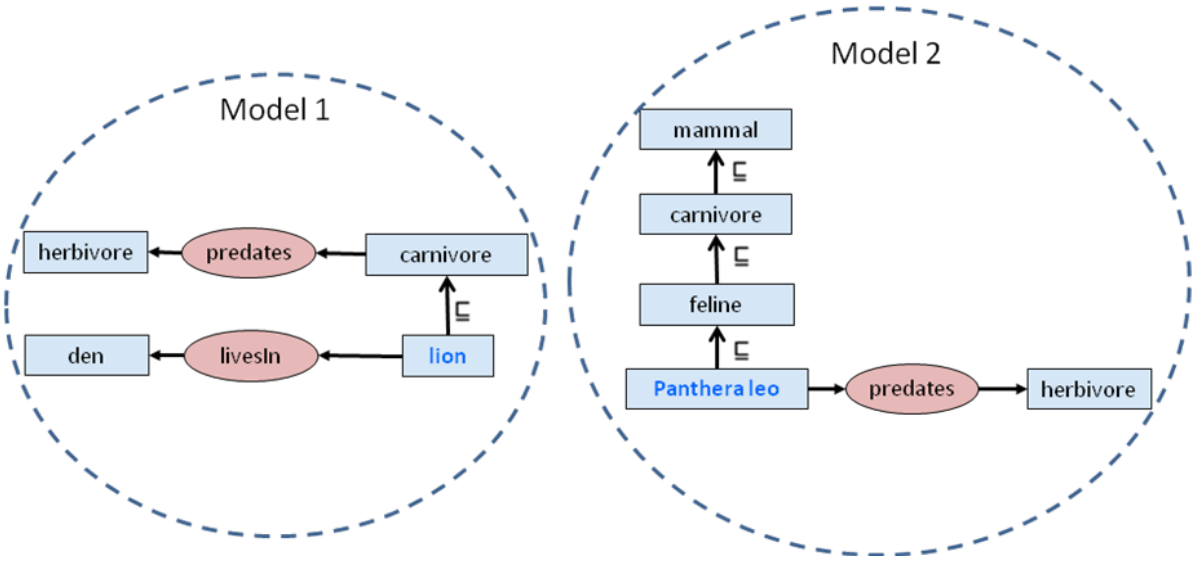

Figure 2: example of models to be matched (⊑ denotes subsumpion)

[3] http://jena.sourceforge.net/

[4] http://jena.sourceforge.net/inference/

If the ontological contexts of “lion” (model 1) and “Panthera leo” (model 2) were extracted without inference, the comparisons to compute the elementary semantic similarities will take place as illustrated in Figure 3, where lexical information of the terms (a) is compared, as well as direct superterms (b) and relationships with other terms (c).

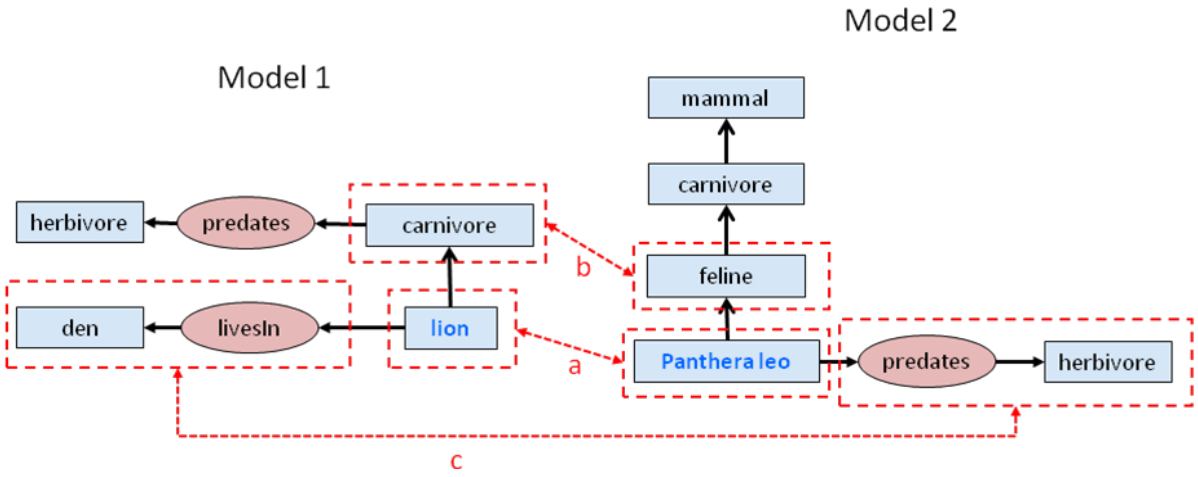

Figure 3: comparisons between the asserted ontologies

Thus, comparisons at levels a, b and c will give several similarity values, that combined by means of the ANN lead to a final similarity value

$$sim_{no\_inference}(model1\#lion, model2\#PantheraLeo)$$

Nevertheless, a simple inspection of the models shows that there is still semantic information in the models not initially considered in the comparisons but that could be reached by applying simple inference rules. For instance, the ancestors of “feline” in model 2 can be reached by transitivity. Furthermore, the fact that “lion” predates “herbivore” (not explicitly declared in model 2) can be reached by applying RDFS inference.

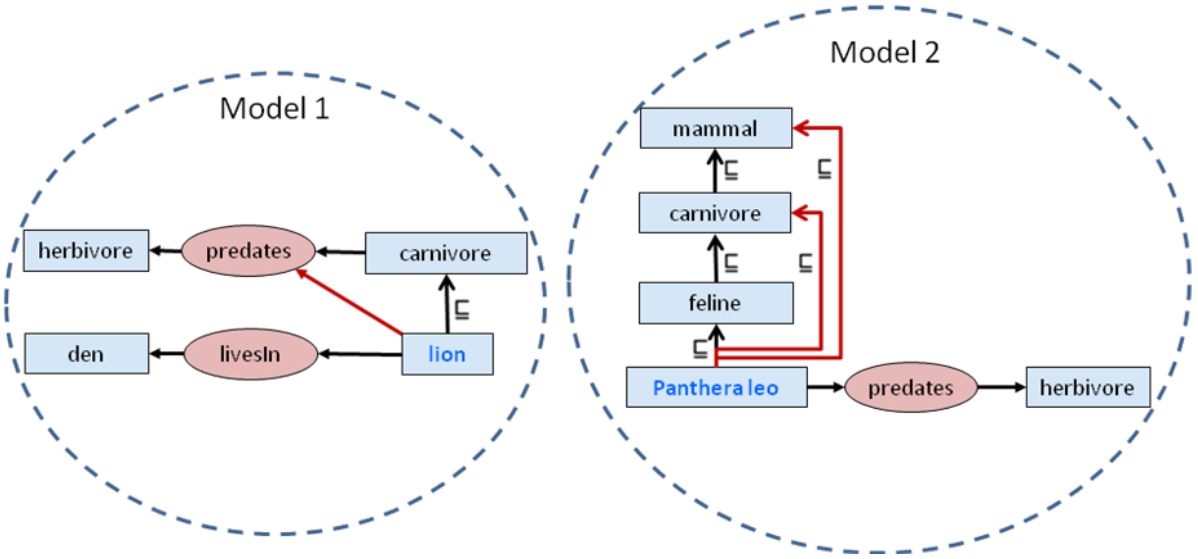

Figure 4: models with new inferred facts

Figure 4 shows the enriched ontological context of “lion” and “Panthera leo” after applying RDFS inference rules. Thus, comparisons between them for computing similarities are richer now and new

common facts can be easily detected (such as the fact that “lion” is a “carnivore” or that it “predates herbivore”). This is schematized in Figure 5.

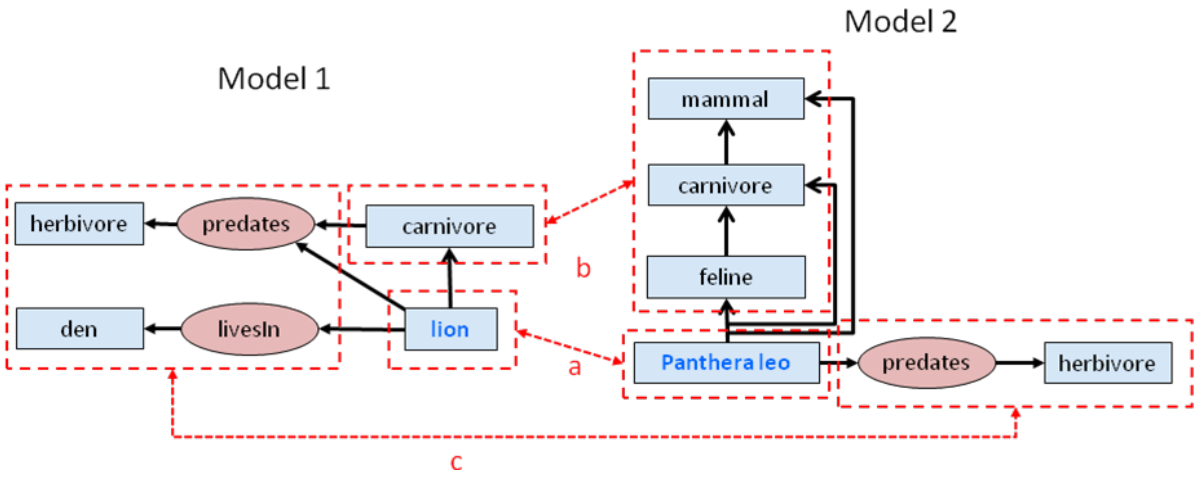

Figure 5: comparisons between the inferred ontologies

Comparisons between the ontological contexts lead to a new similarity value:

$$sim_{RDFS\_inference}(model1\#lion, model2\#PantheraLeo)$$

Owing to the fact that more similar facts are captured in the second case, we have in this case that

$$sim_{no\_inference}(model1\#lion, model2\#PantheraLeo) < sim_{RDFS\_inference}(model1\#lion, model2\#PantheraLeo)$$

The increment in the similarity value makes easier to identify both concepts as equivalent, thus increasing the possibilities of generating useful semantic feedback for that.

3. Inference during semantic feedback

In this section we review the typical errors that modellers can make when representing taxonomical knowledge (inconsistency, incompleteness, etc.). Then, the particularities of modelling in DynaLearn are analysed, which prevents many of the common errors, although limiting also the expressivity of the resultant model. Finally, the rules that The ST component implements to detect some of these inconsistency errors are explained with some detail.

In fact, after the ontology matching process, we use semantic reasoning techniques to detect taxonomic inconsistencies between entities of the two models. As we explained in [10], QR models are treated as ontologies for the generation of semantic feedback. In order to analyse the taxonomic correctness of the model, we follow the categories for taxonomic evaluation proposed in [3], detailed in the following.

3.1. Taxonomic evaluation in ontologies

When representing taxonomic knowledge, there are different types of errors a modeller can make. These errors can be classified in three groups: inconsistency, incompleteness, and redundancy [3]. Figure 6 shows this classification.

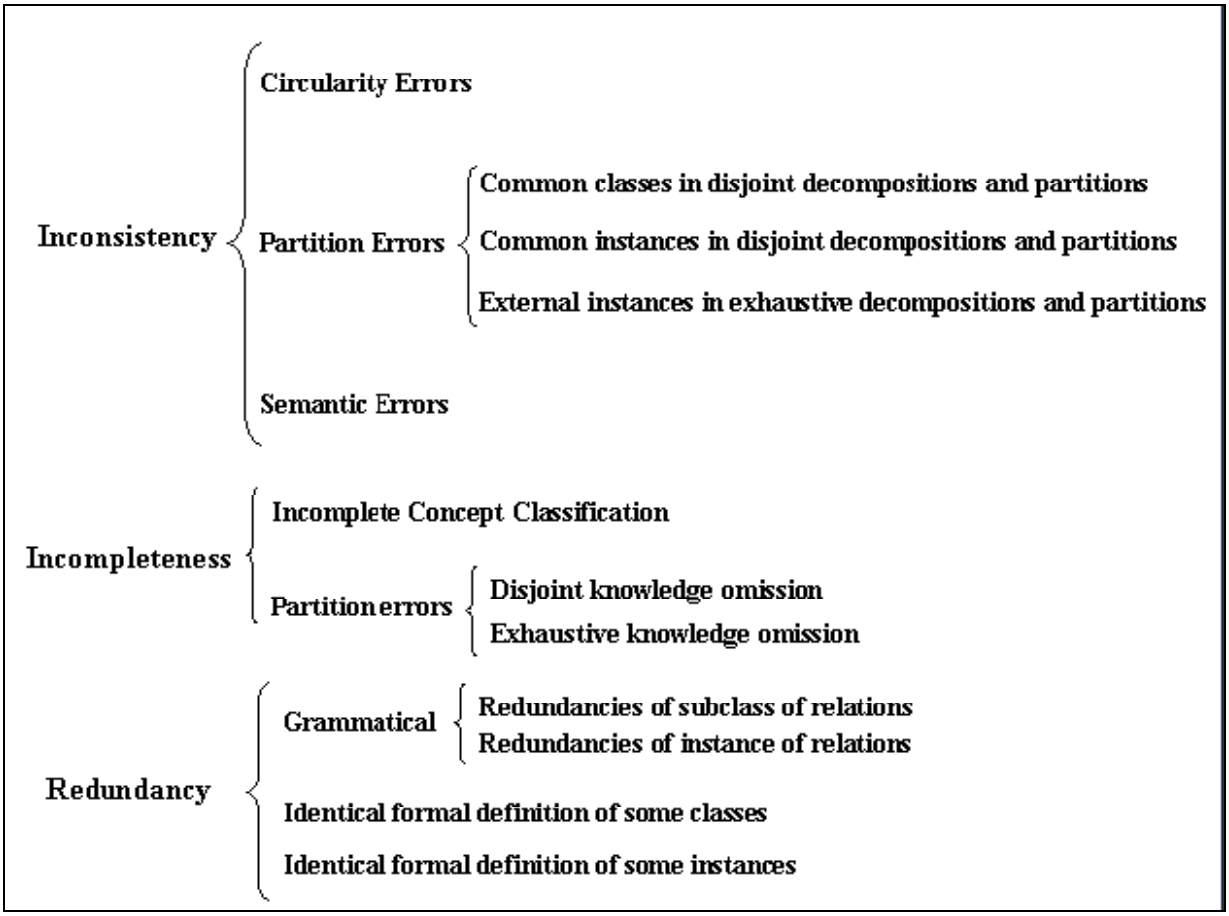

Figure 6: Classification of errors when modelling taxonomies

3.1.1. Inconsistency

Within the inconsistency errors we find three subcategories: circulatory errors, semantic inconsistency errors, and partition errors.

Circulatory errors: They occur when a class is defined as subclass or superclass of itself. This can occur at any level of the hierarchy and with distance 0 (i.e. subclass of itself), 1 (subclass of its superclass) or n. E.g., *lion* isA *feline* and *feline* isA *lion*.

Semantic inconsistency errors: They occur when the modeller makes an incorrect semantic classification and represents a concept as subclass of a concept that it is not really its superclass. E.g., *whale* isA *fish*.

Partition errors: Concept classifications can be categorized in three groups: disjoint (disjoint decompositions), complete (exhaustive decompositions), and disjoint and complete (partitions). In the context of DynaLearn, the taxonomies of QR models can be seen as partitions since the classes are all of them disjoint (the classes do not share common instances). The types of partition errors related to the disjoint decompositions are the following:

- Common classes in disjoint decompositions and partitions. These happen when a class *C* is simultaneously subclass of *A* and *B*, being *A* and *B* defined as disjoint.
- Common instances in disjoint decompositions and partitions. These occur when an instance belongs simultaneously to classes *A* and *B*, being *A* and *B* defined as disjoint.

3.1.2. Incompleteness

Incompleteness on taxonomies occurs when the superclasses of a particular class are imprecise or over-specified, and when explicit information about hierarchical relations is missing. These common omissions can be classified as:

Incomplete concept classification: This happens when the classification of concepts is incomplete and does not cover properly the corresponding domain.

Partition errors: These occur when the definition of disjoint and exhaustive knowledge between classes is omitted. For instance, the modeller omits to define that the subclasses of a given class are disjoint; or the modeller misses the completeness constraint between the subclasses and their superclass.

3.1.3. Redundancy

Opposite to incompleteness, there exist redundancy errors. These occur when there is more than one explicit definition of any hierarchical relation (subclass-of, instance-of, etc.) or when there are two classes or instances with the same formal definition.

3.2. Taxonomic evaluation in DynaLearn

The discussion introduced in Section 3.1 is general for any ontology. Most of the above errors can be automatically detected, in principle, with the intervention of a semantic reasoner (e.g., by classifying

the instances in the taxonomy, membership of an instance to various disjoint classes could be detected). Other modelling errors, such as *incomplete concept classification*, cannot be easily captured by a semantic reasoner and would require expert validation.

Nevertheless, taxonomies in QR models have certain particularities "by construction" that somehow limits the potential of automatic taxonomical validation. The main particularities are:

- All the decompositions in the DynaLearn taxonomies are disjoint.
- Only entities can have an associated hierarchy.
- Multiple inheritances are not allowed.
- Instances cannot be assigned to more than one class.
- Exhaustive decompositions cannot be specified.
- Semantic expressivity is limited. For instance it is no possible to express cardinality restrictions, property restrictions, enumerated classes, Boolean combinations (union, complement, and intersection), etc.

These features are caused by the inherent nature of QR models and the strict controls that the user interface establishes during the creation of the model.

In such scenario, we posed ourselves the following question: *Can semantic reasoning be useful for detecting modelling errors in QR models?* In fact, the above mentioned particularities prevent the user to make most of the errors summarized in Figure 6, at the cost of limiting the semantic expressivity of the model. Therefore, only the following two errors might happen within the scope of a single QR model: incomplete concept classification and inconsistency semantic errors. Only the latter could be detected by semantic reasoners, although with a limited capability due to the lack of class restrictions in the QR models.

Given that situation, *is it still possible to detect taxonomical errors in QR models*? And if so, *how can we do that*? The answer comes from considering not only one model (the one under construction) but considering also the knowledge contained in another reference models. Precisely, comparison with other models is part of the semantic feedback process (see Deliverable D4.2), so the evaluation of taxonomical errors can be performed during that step.

During the generation of semantic feedback, and as result of the ontology matching process, we obtain a list of equivalent terms. Two terms identified as equivalent exhibit some commonalities, like their labels or the resources they are grounded to. In the case of entities at learning space six, they should share not only the terminology but also their hierarchical relations. When the ontological contexts of the matched terms are considered together, it is possible to detect inconsistencies and incompleteness that could not be detected in isolation:

1. Inconsistency -> circularity errors
2. Inconsistency -> partition errors -> common classes and common instances in disjoint decompositions
3. Inconsistency -> semantic errors
4. Incompleteness -> incomplete concept classification
5. Incompleteness -> partition errors -> disjoint knowledge omission

Notice that completeness (exhaustive decompositions) cannot be modelled in DynaLearn, thus being this feature irrelevant from the perspective of giving semantic feedback to users. Also redundancy will not happen due to the constraints imposed by the interface during modelling.

3.2.1. Inconsistency errors

When two models created separately are put in relation by means of the ontology matching techniques, certain semantic inconsistencies can arise. That is the case of *circularity errors* can emerge (e.g., modelA: *animal* isA *livingThing*; model B: *livingThing* isA *Animal*), as well as *partition errors* and *semantic errors*. In order to detect these problems in the learner model, we need to obtain the equivalent classes of the reference model and integrate them into the learner model. Only then we are able to detect the semantic inconsistencies.

As an example, **Figure 7** shows the entity hierarchy of a learner model and **Figure 8** the entity hierarchy of a reference model:

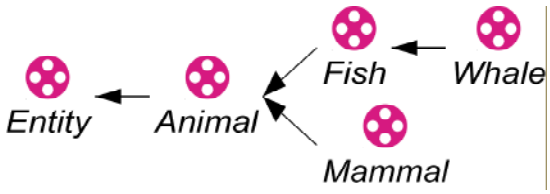

Figure 7: **Entity hierarchy of a learner model**

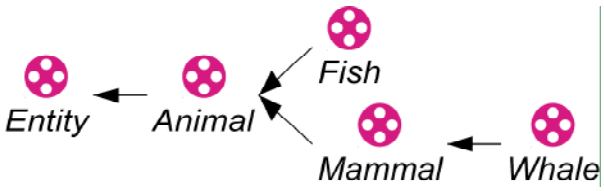

Figure 8: **Entity hierarchy of a reference model**

In this example, we can see that the learner model represents the entity *Whale* as subclass of the entity *Fish*. However, in the reference model another entity *Whale* is defined as subclass of the entity *Mammal*. On the other hand, the entities *Fish* and *Mammal* are disjoint classes, since all classes in these QR models are disjoint by definition. Then, if the entity *Whale* of the learner model has been identified as equivalent to the entity *Whale* of the reference model, we are in front of an inconsistent situation, since the same class cannot be subclass of two disjoint classes.

Algorithm

For each pair of entities found equivalent during the ontology matching process:

1. Get the super class of the term in the reference model (*refSuperclass*).

2. Get the super class of the term in the learner model (*learnerSuperclass*).
3. If both super classes exist, check if the pair of super classes belongs to the list of mappings.
    i. If the super classes are not equivalent, report the inconsistency.
4. Get the subclass of the term in the reference model (*refSubclass*).
5. Get the subclass of the term in the learner model (*learnerSubclass*).
6. If both subclasses exist, check if the pair of subclasses belongs to the list of mappings.
    i. If the subclasses are not equivalent, report the inconsistency.

3.2.2. Incomplete concept classification

Also incompleteness issues can emerge when putting together the learner and reference models. In this case, we need to detect the missing concepts in the hierarchy of the learner model. That is, to find the entities not modelled by the learner but necessary according to the reference model. This process is a particular case for the technique to detect missing terms and that we detailed in Deliverable D4.2. In this case we want to detect the missing entities and, when possible, to find the equivalent position in the learner hierarchy where this missing entity should be added.

4. Inference during instance grounding

In the modelling environment of the DynaLearn software, models can be developed in 6 different learning spaces. Each of these learning spaces is progressively richer in the set of model ingredients that can be used. As such, students can be introduced more gradually to QR concepts. By default, in learning spaces 1 through 5, the software allows model ingredients to be added to the model using a single manipulation. Both of these features are meant to make the software easier to use [8]. For learning space 6, on the contrary, richer modelling techniques are available, enabling the user to explicitly state which concepts are entities and which others are individuals of those entities. Along the remainder of this section we assume that the user operates in learning space 1 to 5.

One consequence of making possible to add model ingredients in a single manipulation is that learners are not explicitly modelling domain concepts. That is, modellers are not explicitly defining the model ingredient definitions representing the vocabulary that is used to compose the rest of the model. These model ingredient definitions (which we will call domain concepts) can be considered to be the classes in an ontology.

In DynaLearn, a learner can create the entity “Water left” to represent the water that is left in a water body in a single manipulation (Figure 9). However, to make it possible to explicitly model domain concepts, the setting “Show types” (which is off by default) can be turned on. When adding a model ingredient, the modeller can indicate both a name for the associated concept (or select a previously defined concept name) and an instance name.

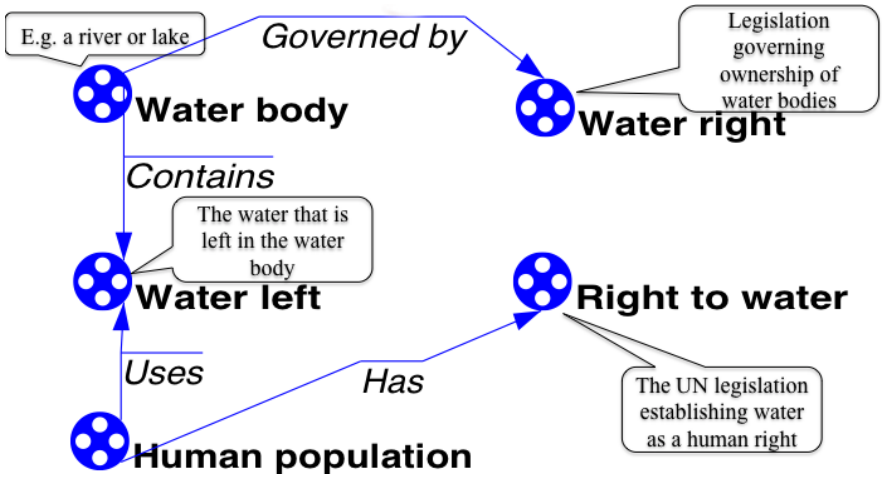

Figure 9: An example model on LS2 created with “Show types” turned off.

When learners will be confronted with the DynaLearn software for the first time, typically, the domain concept names will not be explicitly modelled by students (Show types will be in its default off setting). As such, the models will have an appearance similar to the model shown in Figure 9.

4.1. Identical data structure for “Show types” options

After creating an initial model with “Show types” off, a teacher might decide to encourage the learners to model the domain concepts explicitly. The data structure of the models in DynaLearn have been developed in such a way that by simply turning “Show types” on, the learner can make the domain concepts explicit by adapting the model he was working on. There is no need to start from scratch.

The dialogs that are used to add model ingredients are automatically changes so that the domain concept names can be added. Furthermore, the visualisation of the model is adapted so that the domain concepts are visible (Figure 10).

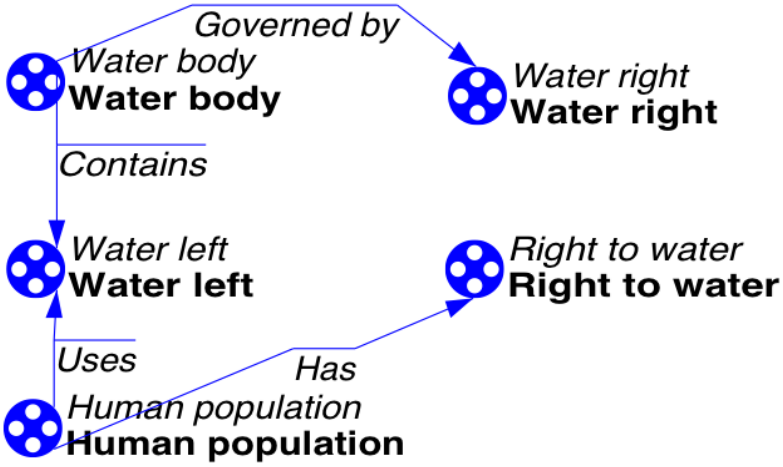

Figure 10: The same model showing the inferred types when "Show types" is turned on.

Technically, the data structure allows such altering of the "Show types" option, by "inferring" the domain concept from the instance name, which is reflected in the software through a model ingredient definition. For example, once the entity "Water left" is added to the model, in the data structure the entity definition "Water left" is automatically created, and the instance "Water left" is made a member of this definition. As such, when the "Show types" option is set to "on" for the model in Figure 9, the visualisation will show Figure 10. The learner is then free to adapt the domain concepts that have been inferred from the instances.

4.2. Problem: "Show types" off issues for grounding and feedback

The interface decision to, by default, not model domain concepts has some consequences on some of the interactions in DynaLearn. It is plausible that learners will never make the step to make the domain concepts explicit. That is, they will keep "Show types" off, and in the model data structure the domain concepts will have been inferred from the names given to the model ingredients. As a result, the domain concepts in these models will be suboptimal.

Models of this type will cause problems particularly for the grounding and semantic feedback functionalities. The grounding functionality attempts to find concepts in DBPedia that match the domain concepts in the QR model [5,6]. There are three issues that make grounding with "Show types" off unusable in its previous incarnation. Firstly, learners modelling with "Show types" off, do not encounter domain concepts in the software during modelling. As such, when they enter the grounding screen, they will encounter the domain concept versions of their instances. If the grounding dialog suggests a domain concept to be renamed, the domain concept will be renamed, but not the instance. This breaks the design choice to keep the domain concepts hidden, and will potentially lead to confusion.

Secondly, when a model contains two instances of the same domain concept (e.g. "container left" and "container right"), they cannot be correctly grounded (to the DBPedia concept "container"). The reason is that DynaLearn prevents different domain concepts to be grounded to the same domain concept (as it would mean that the two domain concepts are equivalent). With "Show types" on, the modeller is encouraged to merge the two domain concepts. However, with "Show types" off, the modeller is not

confronted with domain concepts at all. As such, he does not have the ability to merge the two concepts.

Thirdly and finally, the grounding functionality was not designed to find concepts based on specific examples of such concepts (e.g. the concept “Water” for the water that is left in the water body). As such, the correct results for the grounding can potentially not be found.

The semantic feedback [10,9], which generates suggestions on how to change a particular model based on a large repository of models, is affected due to the models not adequately being grounded. An important step in generating the feedback is determining which models in the repository are relevant to generate the feedback from. In order to do this, models are selected that have groundings in common with the learner model. Consequently, the models developed with “Show types” off and which are not or badly grounded, result in poor semantic feedback being generated, as concepts in different models could be wrongly considered equivalent by the ontology matching tool.

For the reasons mentioned above, we advised against using the grounding and semantic feedback functionalities with “Show types” off. However, from the teachers in DynaLearn there is a strong desire to use the grounding and feedback with “Show types” off, as most of them are running evaluations with beginning students. Moreover, the grounding and semantic feedback could have bigger impact on the education if they can also be used with “Show types” off.

4.3. Allowing grounding of instances

Our solution to the issues with grounding and semantic feedback that were discussed in the previous section is to allow instances to be grounded individually. This decision has wide-ranging consequences as will be discussed in the following sections. Most notable to the modellers is the changed grounding dialog that is shown when “Show types” is turned off. This grounding dialog for instances is almost equivalent to the grounding dialog for domain concepts. The main difference is that the list on the left hand side of the screen shows the instances of particular domain concepts instead of the domain concepts themselves (Figure 11).

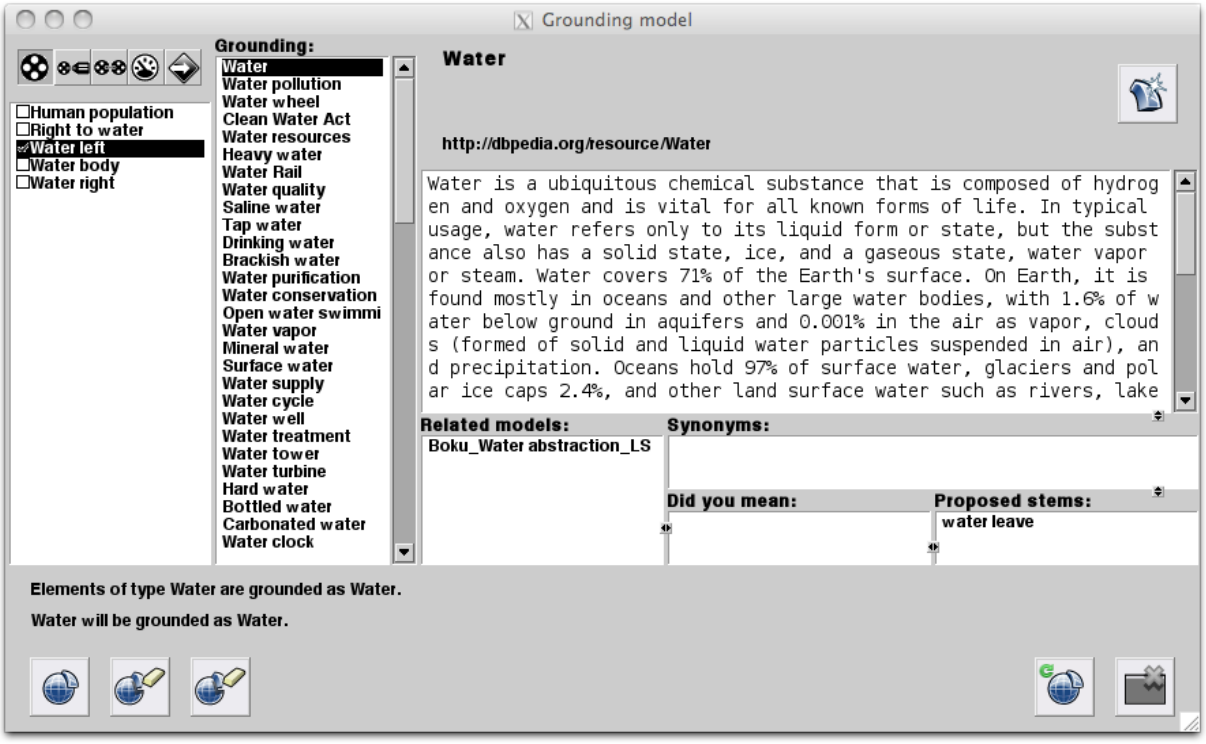

Figure 11: The grounding dialog adapted for instances.

In order to solve the issue of finding domain concepts when grounding using instance names the multi-word grounding functionality is used. By grounding subsets of the words used to describe the instance, the results returned when grounding instances is improved [6]. Since most of this functionality was already in place, it could be easily utilized for instance grounding.

4.4. Solution: Deriving domain concepts from DBPedia concepts

The main difficulties of grounding instances, namely grounding multiple instances of the same domain concept and not confronting modellers with domain concepts (with “Show types” off) are resolved by deriving domain concepts from the DBPedia concepts that the instances are grounded to. The example that inspires this idea is simple. Suppose we have both “Container left” and “Container right” in a model. If a modeller grounds “Container left” to the DBPedia concept “Container”, we can use this DBPedia label to improve the label of the domain concept in DynaLearn by renaming “Container left” to “Container”. The instance name is preserved as “Container left”. As such, the modeller is unaware that this improvement of the domain concepts has occurred, as is the goal with “Show types” turned off.

When “Container right” is also grounded to DBPedia “Container”, a clash occurs. DynaLearn does not allow multiple domain concepts to be grounded to the same concept, as it has a unique name assumption (things with different names refer to different concepts). To resolve this issue, “Container right” is made an instance of “Container” (formally “Container left”), and “Container right” is deleted. As a result, the model is correctly grounded, and both instances are part of the same domain concept. Moreover, the learner is kept unaware of that these changes have occurred and is kept unaware of domain concepts (as is the design choice with “Show types off”). As a result of the correct grounding, the semantic feedback can be correctly used.

The procedure described above should give a good indication of the form of the solution. However, a complete solution is not as simple as the example suggests. For example, when the grounding of one

of the instances for which the domain concepts were merged is deleted (e.g. for “Container right”), the original domain concept has to be restored. A more complex example is when a grounded instance with a merged domain concept is grounded to another grounded instance, e.g. “Water right” which is grounded to DBPedia “Water” is grounded to DBPedia “Water right” (the legislation governing water bodies). First, the domain concept of water has to be unmerged, which is equivalent to deleting the grounding for “Water right”. Secondly, the instance “Water right” has to be made an instance of the already existing definition of “Water right” and its original definition has to be deleted. At the same time, name clashes for domain concepts have to be prevented.

The complete procedure to allow instance grounding to infer the correct domain concepts (including making sure the instances belong to the correct domain concept) is shown in Figure 12 and Figure 13 (note that the domain concepts are called “instance definition” in the images). This procedure uses the grounding process to improve the domain concepts using the labels in DBPedia. As a result, the instances are members of the correct domain concept and the domain concepts are named correctly. Finally, in addition to making the grounding work with “Show types” off, it also makes the use of the semantic feedback functionality possible with this setting.

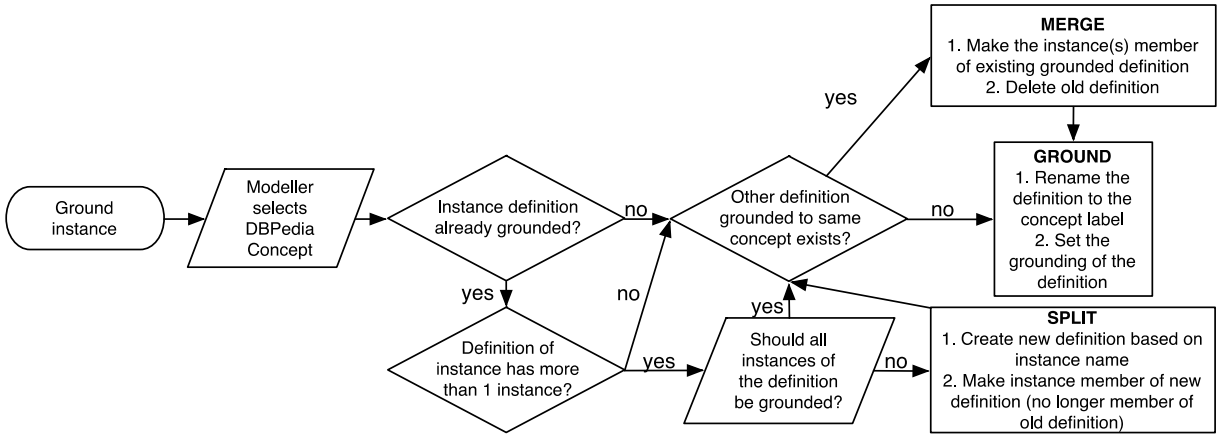

Figure 12: Flow diagram showing the procedure followed when grounding instances.

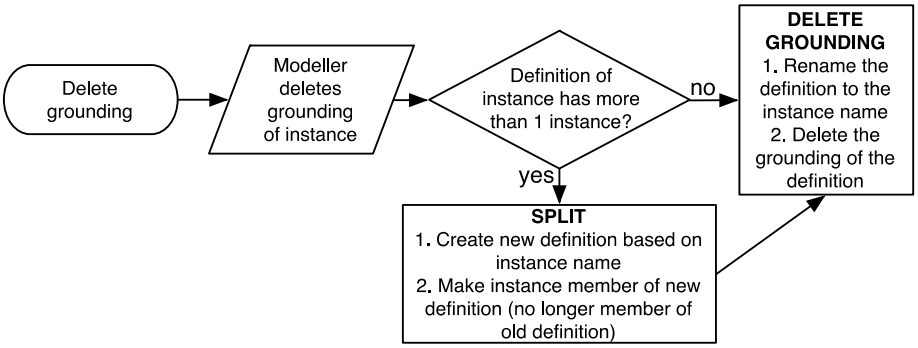

Figure 13: Flow diagram showing the procedure when deleting groundings of instances.

5. Conclusion

This deliverable describes the progress on Task 4.4 “Integrate taxonomic reasoning” aimed at improving the quality of the semantic information presented to users, since taxonomic reasoning “will allow relating topics that would not be considered as similar initially”.

First, we have discussed the opportunities of discovering better mappings between models during ontology matching by applying lightweight reasoning techniques. Second, a discussion of the typical modelling error when representing taxonomical knowledge has been presented, and how they are influenced by the particularities of DynaLearn. As a consequence of this analysis, different rules to detect inconsistencies during semantic feedback have been presented. Finally, we have analysed how the simplified way in which DynaLearn deals with individuals in learning spaces 1 to 5 leads to a suboptimal way of naming domain concepts. We have presented a solution to this issue that takes advantage of the grounding process to infer better concept names.

6. Discussion

Although taxonomical reasoning was initially planned in the project in the context of collaborative filtering, we have discovered several other aspects in our algorithms that may benefit of that, namely: ontology matching, semantic feedback generation, and grounding, thus resulting in a better semantic feedback and recommendation as we have discussed in this deliverable.

Nevertheless, some of the ideas contained in this document still need to be further tested and evaluated. In fact, we plan to perform more tests during the remainder of the project in parallel with the evaluations carried out for WP7. That will allow us to further test our hypothesis and figure out new ways of enriching our techniques in the future.

References

1. D. Billsus, C.A. Brunk, C. Evans, B. Gladish, and M. Pazzani (2002). “Adaptive interfaces for ubiquitous web access”, Communications of the ACM, 45(5): 34-38.

2. J. Euzenat, “An API for ontology alignment” in 3rd International Semantic Web Conference (ISWC'04), Hiroshima (Japan). Springer, November 2004.

3. A. Gómez-Pérez. “Evaluation of Ontologies”, International Journal of Intelligent Systems 16(3):391-409.

4. J. Gracia, J. Bernad and E. Mena, “Ontology matching with CIDER: Evaluation report for OAEI 2011”, in Proc. of 6rd Ontology Matching Workshop (OM'11), at ISWC'11, Bonn, Germany, CEUR-WS, October 2011.

5. J. Gracia, J. Liem, E. Lozano, O. Corcho, M. Trna, A. Gómez-Pérez, and B. Bredeweg, "Semantic techniques for enabling knowledge reuse in conceptual modelling," in Proc. of 9th International Semantic Web Conference (ISWC2010), Shanghai(China), ser. LNCS, vol. 6497. Springer, Nov. 2010, pp. 82-97.

6. J. Gracia, M. Trna, E. Lozano, T.T. Nguyen, A. Gómez-Pérez, C. Montaña, and J. Liem, “Semantic repository and ontology mapping”. DynaLearn, EC FP7 STREP project 231526, Deliverable D4.1. 2010.

7. P. Hayes, "RDF semantics," W3C, Tech. Rep., Feb. 2004. Available: http://www.w3.org/TR/rdf-mt/

8. J. Liem, W. Beek, and B. Bredeweg. “Differentiating Qualitative Representations into Learning Spaces”. 24th International Workshop on Qualitative Reasoning (QR’10), Portland, Oregon, USA, pages 37-46, August 2010.

9. E. Lozano et al. “Model-based and memory-based collaborative filtering algorithms for complex knowledge models”. DynaLearn, EC FP7 STREP project 231526, Deliverable D4.3. 2011

10. E. Lozano, J. Gracia, A. Gómez-Pérez, J. Liem, C. van Weelden, and B. Bredeweg. “Ontology-based feedback on model quality.” DynaLearn, EC FP7 STREP project 231526, Deliverable D4.2. 2010

11. E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF," W3C Recommendation, Tech. Rep., Jan. 2008. Available at: http://www.w3.org/TR/rdf-sparql-query/

12. M. Smith. “Neural Networks for Statistical Modeling,” John Wiley & Sons, Inc., New York, NY, USA, 1993.

13. V. Raghavan and S. Wong. “A critical analysis of vector space model for information retrieval”. In Journal of the American society for information science 37, 1986.

Appendix A: Semantic Technologies Testing Plan

A1. Overview

This section describes the testing plan executed on a weekly basis by the UPM team as part of the DynaLearn Testing Procedures. The *Scope* section describes the major functionalities included in the test plan, the *Testing Process* section explains the initial conditions, the process steps that are automatically executed and the final results. At the end of this Appendix the complete *test cases* list is presented.

A2. Scope

The following table lists the major functionalities that are included in the test process. They are represented as test suites that contain several test cases covering and ensuring the proper operation of the application. The complete list of test cases is described in “Test Cases Definition” section.

Test Suite	**Description**
CUS001: Log in to repository	Validates the authentication service to the semantic repository.
CUS002: Open model	Validates the correct reception of a model from the repository.
CUS003: Storage model	Validates the correct saving process of models in the semantic repository.
CUS004: Ground model	Validates the grounding of entire modules functionality. Models in English, Spanish, Portuguese and German are tested.
CUS005: Ground term	Validates the grounding of term functionality. Single and Compound words are used in the test as well as different languages: English, Spanish, Portuguese and German.
CUS007: List models	Validates the list models functionality.
CUS008: Query models	Validates the model filtering by using a term.
CUS009: Update model	Validates the replacing of an existing model into repository
CUS010: Open HGP model	Validates the correct reception of an available model in repository in HGP binary format.
CUS011: Get Feedback	Validates the feedback functionality service.

A3. Testing Process

The following initial conditions are required before executing the test:

- **Accounts**: To execute the test plan the following valid users and passwords are required.
  - demo-student@dynalearn.eu
  - demo-teacher@dynalearn.eu
  - upm-domex@example.com
- **Test Server**: Testing will be done on testing web service http://elnath.dia.fi.upm.es:8020/SemanticTechnology/services/WSServerSecured?wsdl
- **Models**: The following are the HGP models used in this test plan
  - CommunicatingVessels_testing.hgp
  - GroundingInSpanish.hgp
  - GroundingInPortuguese.hgp
  - GoungdingInGerman.hgp
  - Example_RefModel.hgp

The test is automatically executed by using ANT, SoapUI and a script that exports the models into OWL. Find below the flow chart describing the automatic process performed during the test execution:

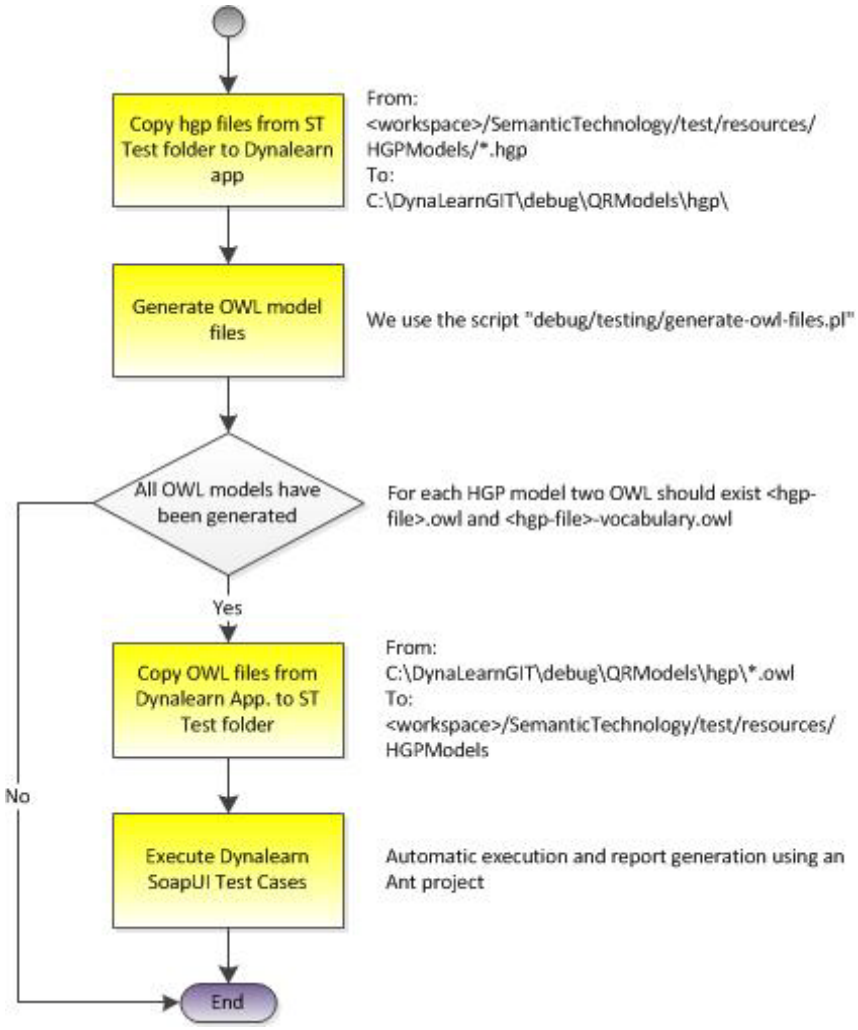

Figure 14: Flowchart of the testing process

The testing results and reports are automatically generated too, these include test suite results, failed test case reports, and a complete process report in HTML format; the following screenshots provides a better idea of the final report:

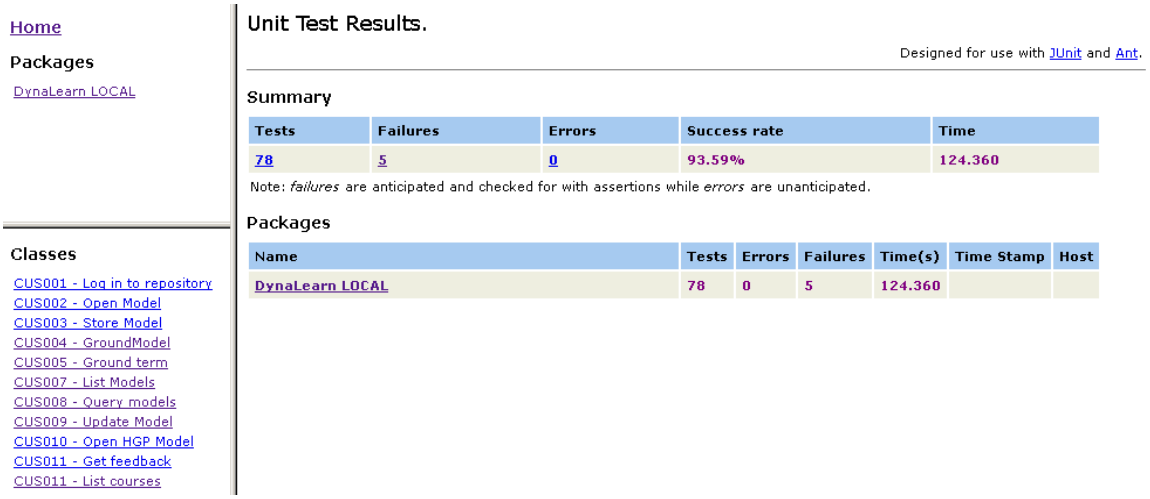

Figure 15: Global view of the test report

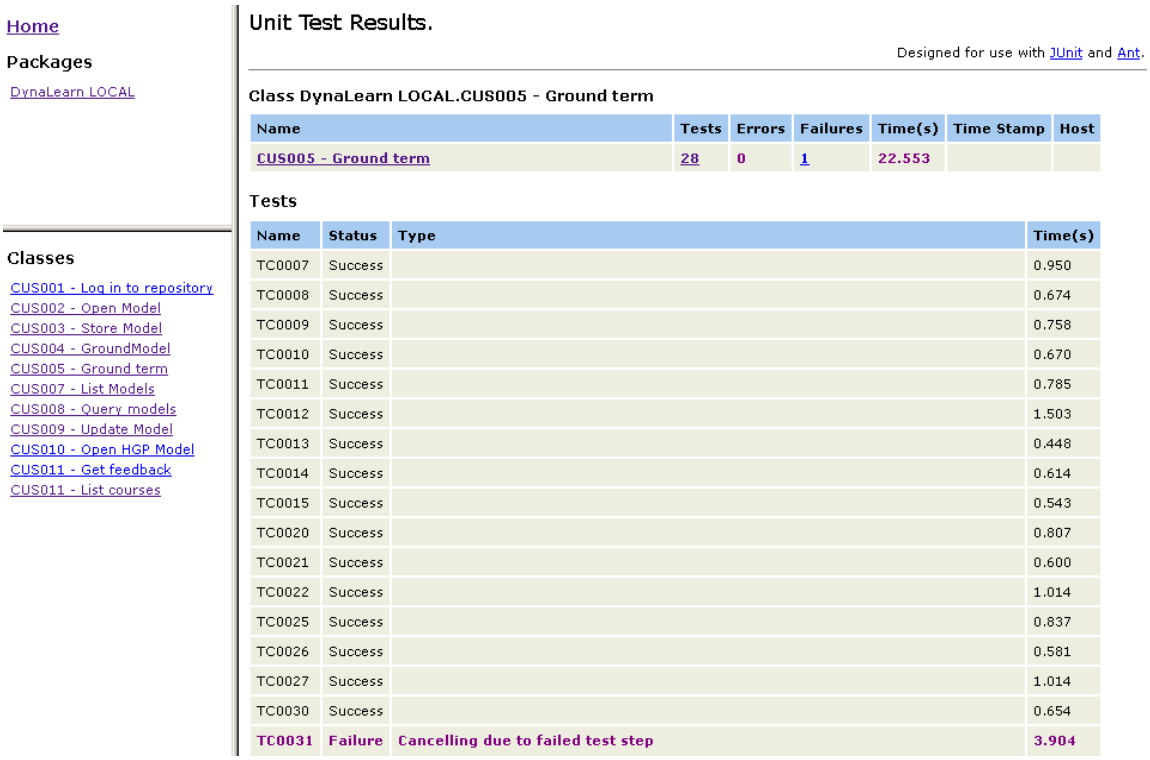

Figure 16: Test suite details

A *failure* happens when one of the test case assertions fails. That is, the program does something wrong and the test case notices and reports the fact. An *error* occurs when some other unexpected Exception is triggered (e.g., a NullPointerException or an ArrayIndexOutOfBoundsException).

A4. Test Cases Definition

Each Test Suite is tested in different ways. For doing this, there are some test cases associated to each test suite. This section specifies each automatic test case defined to ensure the availability of the ST in DynaLearn project.

In order to understand the specifications below, the fields are defined in the next list:

- Description: Brief description of the test case.
- Role: Role of the user who is calling the service for testing.
- Use case: The test suite to which a test case is related.
- Scenario: A test case could have one of two possible values in this field (correct and incorrect) whenever the input is a right or wrong value. In both cases the system must work.
- Term type: During grounding, the term to ground could be single o multi word.
- Grounding type: During grounding, the term can be grounded to DBpedia (or any other background ontology) or grounded to an anchor term.
- Ingredient: It refers to the ingredient to be grounded in the QR model, for example: entity, agent, configuration, etc.
- Language: The language of the involved term or model.
- Input: Web method calling parameter.
- Web method: Web method to call.

TEST CASE: TC0001

Description:	A simple correct login test by using teacher role		
Role:	teacher	**Use case:**	CUS001_Log_in_to_repository
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient**:	
Language:		**Input:**	demo-teacher@dynalearn.eu, te@cher
Web method:			

TEST CASE: TC0002

Description:	A simple incorrect login test by using teacher role		
Role:	teacher	**Use case:**	CUS001_Log_in_to_repository
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	demo-teacher@dynalearn.eu, xxxxx
Web method:			

TEST CASE: TC0003

Description:	A simple correct login test by using learner role		
Role:	learner	**Use case:**	CUS001_Log_in_to_repository
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	demo-student@dynalearn.eu,student
Web method:			

TEST CASE: TC0004

Description:	A simple incorrect login test by using learner role		
Role:	learner	**Use case:**	CUS001_Log_in_to_repository

Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	demo-student@dynalearn.eu, xxxx
Web method:			

TEST CASE: TC0005

Description:	A simple correct login test by using domain expert role		
Role:	domain expert	**Use case:**	CUS001_Log_in_to_repository
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	upm-domex@example.com, UPMDOMEX
Web method:			

TEST CASE: TC0006

Description:	A simple incorrect login test by using domain expert role		
Role:	domain expert	**Use case:**	CUS001_Log_in_to_repository
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	upm-domex@example.com, xxxx
Web method:			

TEST CASE: TC0007

Description:	Grounding of a correct single word english term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	english	**Input:**	Labour
Web method:	groundTerm		

TEST CASE: TC0008

Description:	Grounding of a correct single word english term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Agent
Language:	english	**Input:**	Biochemistry
Web method:	groundTerm		

TEST CASE: TC0009

Description:	Grounding of a correct single word english term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Quantity

Language:	english	**Input:**	Leakage
Web method:	groundTerm		

TEST CASE: TC0010

Description:	Grounding of a correct single word english term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Configuration
Language:	english	**Input:**	Ecosystem
Web method:	groundTerm		

TEST CASE: TC0011

Description:	Grounding of a correct single word english term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Attribute
Language:	english	**Input:**	Dead
Web method:	groundTerm		

TEST CASE: TC0012

Description:	Grounding of a correct multi word english term. It also includes a test of suggested terms.		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	multi word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	english	**Input:**	Heat_transfer
Web method:	groundTerm		

TEST CASE: TC0013

Description:	Grounding of a incorrect single word english term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	incorrect	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	XXX
Language:	english	**Input:**	Renewable_resources
Web method:	groundTerm		

TEST CASE: TC0014

Description:	Grounding of a incorrect single word english term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	incorrect	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	english	**Input:**	IndustrialxxXXxx

Web method:	groundTerm

TEST CASE: TC0015

Description :	Grounding of a incorrect multi word english term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	incorrect	**Term type:**	multi word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	english	**Input:**	Energy_pyramidxx
Web method:	groundTerm		

TEST CASE: TC0016

Description :	Grounding of a correct multi word english term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	multi word
Grounding type:	anchor term	**Ingredient:**	
Language:	english	**Input:**	Climate_regulation
Web method:	createAnchorTerm		

TEST CASE: TC0017

Description :	Grounding of a correct single word english term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	anchor term	**Ingredient:**	
Language:	english	**Input:**	Allelopathy
Web method:	createAnchorTerm		

TEST CASE: TC0018

Description :	Grounding of a incorrect single word english term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	incorrect	**Term type:**	single word
Grounding type:	anchor term	**Ingredient:**	
Language:	english	**Input:**	Industrialxx
Web method:	createAnchorTerm		

TEST CASE: TC0019

Description :	Grounding of a incorrect multi word english term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	incorrect	**Term type:**	multi word
Grounding type:	anchor term	**Ingredient:**	
Language:	english	**Input:**	Energy_pyramidxx
Web method:	createAnchorTerm		

TEST CASE: TC0020

Description :	Grounding of a correct single word portuguese term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	portuguese	**Input:**	água
Web method:	groundTerm		

TEST CASE: TC0021

Description :	Grounding of a incorrect single word portuguese term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	incorrect	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	portuguese	**Input:**	Renewable_resources
Web method:	groundTerm		

TEST CASE: TC0022

Description :	Grounding of a correct multi word portuguese term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	multi word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	portuguese	**Input:**	Fluxo german calor
Web method:	groundTerm		

TEST CASE: TC0023

Description :	Grounding of a correct single word portuguese term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	anchor term	**Ingredient:**	
Language:	portuguese	**Input:**	Desmatamento
Web method:	createAnchorTerm		

TEST CASE: TC0024

Description :	Grounding of a correct multi word portuguese term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	multi word
Grounding type:	anchor term	**Ingredient:**	
Language:	portuguese	**Input:**	Contaminantes do solo
Web method:	createAnchorTerm		

TEST CASE: TC0025

Description :	Grounding of a correct single word spanish term

Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	spanish	**Input:**	café
Web method:	groundTerm		

TEST CASE: TC0026

Description :	Grounding of a incorrect single word spanish term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	incorrect	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	spanish	**Input:**	Renewable_resources
Web method:	groundTerm		

TEST CASE: TC0027

Description :	Grounding of a correct multi word spanish term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	multi word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	spanish	**Input:**	Flujo german calor
Web method:	groundTerm		

TEST CASE: TC0028

Description :	Grounding of a correct single word spanish term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	anchor term	**Ingredient:**	
Language:	spanish	**Input:**	Deforestación
Web method:	createAnchorTerm		

TEST CASE: TC0029

Description :	Grounding of a correct multi word spanish term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	multi word
Grounding type:	anchor term	**Ingredient:**	
Language:	spanish	**Input:**	Contaminantes del suelo
Web method:	createAnchorTerm		

TEST CASE: TC0030

Description :	Grounding of a correct single word german term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word

Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	german	**Input:**	Wasser
Web method:	groundTerm		

TEST CASE: TC0031

Description :	Grounding of a incorrect single word german term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	incorrect	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	german	**Input:**	Renewable_resources
Web method:	groundTerm		

TEST CASE: TC0032

Description :	Grounding of a correct multi word german term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	multi word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	german	**Input:**	Wärmeleitfähigkeit
Web method:	groundTerm		

TEST CASE: TC0033

Description :	Grounding of a correct single word german term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	anchor term	**Ingredient:**	
Language:	german	**Input:**	Entwaldung
Web method:	createAnchorTerm		

TEST CASE: TC0034

Description :	Grounding of a correct multi word german term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	multi word
Grounding type:	anchor term	**Ingredient:**	
Language:	german	**Input:**	Weißes Haus
Web method:	createAnchorTerm		

TEST CASE: TC0040

Description :	Grounding of a correct single word dutch term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	dutch	**Input:**	Vuur
Web method:	groundTerm		

TEST CASE: TC0041

Description :	Grounding of a incorrect single word dutch term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	incorrect	**Term type:**	single word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	dutch	**Input:**	Renewable_resources
Web method:	groundTerm		

TEST CASE: TC0042

Description :	Grounding of a correct multi word dutch term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	multi word
Grounding type:	dbpedia	**Ingredient:**	Entity
Language:	dutch	**Input:**	Warmteoverdracht
Web method:	groundTerm		

TEST CASE: TC0043

Description :	Grounding of a correct single word dutch term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	single word
Grounding type:	anchor term	**Ingredient:**	
Language:	dutch	**Input:**	Acteur
Web method:	createAnchorTerm		

TEST CASE: TC0044

Description :	Grounding of a correct multi word dutch term		
Role:		**Use case:**	CUS005_Ground_term
Scenario:	correct	**Term type:**	multi word
Grounding type:	anchor term	**Ingredient:**	
Language:	dutch	**Input:**	Atmosferisch perspectief
Web method:	createAnchorTerm		

TEST CASE: TC0045

Description :	Grounding of an entire english model		
Role:		**Use case:**	CUS004_Ground_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	english	**Input:**	CommunicatingVessels_testing.owl
Web method:	groundModel		

TEST CASE: TC0046

Description :	Grounding of an entire portuguese model		
Role:		**Use case:**	CUS004_Ground_model

Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	portuguese	**Input:**	GroundingInPortuguese.owl
Web method:	groundModel		

TEST CASE: TC0047

Description :	Grounding of an entire spanish model		
Role:		**Use case:**	CUS004_Ground_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	spanish	**Input:**	GroundingInSpanish.owl
Web method:	groundModel		

TEST CASE: TC0048

Description :	Grounding of an entire german model		
Role:		**Use case:**	CUS004_Ground_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	german	**Input:**	GoungdingInGerman.owl
Web method:	groundModel		

TEST CASE: TC0050

Description :	Grounding of an entire dutch model		
Role:		**Use case:**	CUS004_Ground_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	dutch	**Input:**	GroundingInDutch.owl
Web method:	groundModel		

TEST CASE: TC0051

Description :	Store of an entire english model into repository		
Role:	teacher	**Use case:**	CUS003_Save_model_to_repository
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	english	**Input:**	CommunicatingVessels_testing.owl
Web method:	storeModel		

TEST CASE: TC0053

Description :	Store of an entire portuguese model into repository		
Role:	learner	**Use case:**	CUS003_Save_model_to_repository
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	portuguese	**Input:**	GroundingInPortuguese.owl

Web method:	storeModel

TEST CASE: TC0055

Description :	Store of an entire spanish model into repository		
Role:	domain expert	**Use case:**	CUS003_Save_model_to_repository
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	spanish	**Input:**	GroundingInSpanish.owl
Web method:	storeModel		

TEST CASE: TC0052

Description :	Store of an entire german model into repository		
Role:	teacher	**Use case:**	CUS003_Save_model_to_repository
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	german	**Input:**	GoungdingInGerman.owl
Web method:	storeModel		

TEST CASE: TC0056

Description :	Store of an entire dutch model into repository		
Role:	domain expert	**Use case:**	CUS003_Save_model_to_repository
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	dutch	**Input:**	GroundingInDutch.owl
Web method:	storeModel		

TEST CASE: TC0057

Description :	correct retrieval of a model from repository		
Role:	teacher	**Use case:**	CUS002_Open_model_from_repository
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	CommunicatingVessels_testing.owl
Web method:	retrieveModel		

TEST CASE: TC0058

Description :	correct retrieval of a model from repository		
Role:	teacher	**Use case:**	CUS002_Open_model_from_repository
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	GroundingInPortuguese.owl
Web method:	retrieveModel		

TEST CASE: TC0059

Description :	correct retrieval of a model from repository

Role:	learner	**Use case:**	CUS002_Open_model_from_repository
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	GroundingInSpanish.owl
Web method:	retrieveModel		

TEST CASE: TC0060

Description :	incorrect retrieval of a model from repository		
Role:	learner	**Use case:**	CUS002_Open_model_from_repository
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	xxCommunicatingVessels_testing
Web method:	retrieveModel		

TEST CASE: TC0061

Description :	incorrect retrieval of a model from repository		
Role:	domain expert	**Use case:**	CUS002_Open_model_from_repository
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	xxCommunicatingVessels_testing
Web method:	retrieveModel		

TEST CASE: TC0062

Description :	incorrect retrieval of a model from repository		
Role:	domain expert	**Use case:**	CUS002_Open_model_from_repository
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	xxCommunicatingVessels_testing
Web method:	retrieveModel		

TEST CASE: TC0069

Description :	list models using teacher role		
Role:	teacher	**Use case:**	CUS007_list_models
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	
Web method:	listModels		

TEST CASE: TC0070

Description :	list models using learner role		
Role:	learner	**Use case:**	CUS007_list_models
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	

Language:		**Input:**	
Web method:	listModels		

TEST CASE: TC0071

Description :	list models using domain expert role		
Role:	domain expert	**Use case:**	CUS007_list_models
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	
Web method:	listModels		

TEST CASE: TC0172

Description :	list models using teacher role and selecting models related to a course		
Role:	teacher	**Use case:**	CUS007_list_models
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	uva-crse
Web method:	listModels		

TEST CASE: TC0072

Description :	correct query english model		
Role:	teacher	**Use case:**	CUS008_query_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	english	**Input:**	Container
Web method:	queryModels		

TEST CASE: TC0073

Description :	incorrect query english model		
Role:	teacher	**Use case:**	CUS008_query_model
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	english	**Input:**	Containerxxx
Web method:	queryModels		

TEST CASE: TC0074

Description :	correct query english model		
Role:	learner	**Use case:**	CUS008_query_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	english	**Input:**	Container
Web method:	queryModels		

TEST CASE: TC0075

Description :	incorrect query english model		
Role:	learner	**Use case:**	CUS008_query_model
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	english	**Input:**	Containerxxx
Web method:	queryModels		

TEST CASE: TC0076

Description :	correct query english model		
Role:	domain expert	**Use case:**	CUS008_query_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	english	**Input:**	Container
Web method:	queryModels		

TEST CASE: TC0077

Description :	incorrect query english model		
Role:	domain expert	**Use case:**	CUS008_query_model
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	english	**Input:**	Containerxxx
Web method:	queryModels		

TEST CASE: TC0178

Description :	correct query portuguese model		
Role:	teacher	**Use case:**	CUS008_query_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	portuguese	**Input:**	adulto
Web method:	queryModels		

TEST CASE: TC0179

Description :	incorrect query portuguese model		
Role:	teacher	**Use case:**	CUS008_query_model
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	portuguese	**Input:**	Containerxxx
Web method:	queryModels		

TEST CASE: TC0180

Description :	correct query spanish model		
Role:	teacher	**Use case:**	CUS008_query_model
Scenario:	correct	**Term type:**	

Grounding type:		**Ingredient:**	
Language:	spanish	**Input:**	Cerveza
Web method:	queryModels		

TEST CASE: TC0181

Description :	incorrect query spanish model		
Role:	teacher	**Use case:**	CUS008_query_model
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	spanish	**Input:**	Containerxxx
Web method:	queryModels		

TEST CASE: TC0182

Description :	correct query german model		
Role:	teacher	**Use case:**	CUS008_query_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	german	**Input:**	akne
Web method:	queryModels		

TEST CASE: TC0183

Description :	incorrect query german model		
Role:	teacher	**Use case:**	CUS008_query_model
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	german	**Input:**	Containerxxx
Web method:	queryModels		

TEST CASE: TC0078

Description :	Update of an entire english model into repository		
Role:	teacher	**Use case:**	CUS009_update_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	english	**Input:**	CommunicatingVessels_testing.owl
Web method:	storeModel		

TEST CASE: TC0079

Description :	Update of an entire portuguese model into repository		
Role:	learner	**Use case:**	CUS009_update_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	portuguese	**Input:**	GroundingInPortuguese.owl
Web method:	storeModel		

TEST CASE: TC0080

Description :	Update of an entire spanish model into repository		
Role:	domain expert	**Use case:**	CUS009_update_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	spanish	**Input:**	GroundingInSpanish.owl
Web method:	storeModel		

TEST CASE: TC0081

Description :	Update of an entire german model into repository		
Role:	teacher	**Use case:**	CUS009_update_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	german	**Input:**	GoungdingInGerman.owl
Web method:	storeModel		

TEST CASE: TC0083

Description :	Update of an entire dutch model into repository		
Role:	domain expert	**Use case:**	CUS009_update_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:	dutch	**Input:**	GroundingInDutch.owl
Web method:	storeModel		

TEST CASE: TC0084

Description :	correct retrieval of a binary hgp model from repository		
Role:	teacher	**Use case:**	CUS010_retrieve_hgp_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	CommunicatingVessels_testing.owl
Web method:	retrieveModel		

TEST CASE: TC0085

Description :	correct retrieval of a binary hgp model from repository		
Role:	learner	**Use case:**	CUS010_retrieve_hgp_model
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	GroundingInPortuguese.owl
Web method:	retrieveModel		

TEST CASE: TC0086

Description :	correct retrieval of a binary hgp model from repository		
Role:	domain expert	**Use case:**	CUS010_retrieve_hgp_model

Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	GroundingInSpanish.owl
Web method:	retrieveModel		

TEST CASE: TC0087

Description :	incorrect retrieval of a binary hgp model from repository		
Role:	teacher	**Use case:**	CUS010_retrieve_hgp_model
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	xxCommunicatingVessels_testing
Web method:	retrieveModel		

TEST CASE: TC0088

Description :	incorrect retrieval of a binary hgp model from repository		
Role:	learner	**Use case:**	CUS010_retrieve_hgp_model
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	xxCommunicatingVessels_testing
Web method:	retrieveModel		

TEST CASE: TC0089

Description :	incorrect retrieval of a binary hgp model from repository		
Role:	domain expert	**Use case:**	CUS010_retrieve_hgp_model
Scenario:	incorrect	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	xxCommunicatingVessels_testing
Web method:	retrieveModel		

TEST CASE: TC0201

Description :	list courses using teacher role		
Role:	teacher	**Use case:**	CUS011_list_courses
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	
Web method:	listCourses		

TEST CASE: TC0202

Description :	list courses using learner role		
Role:	learner	**Use case:**	CUS011_list_courses
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	

Web method:	listCourses

TEST CASE: TC0203

Description :	list courses using domain expert role		
Role:	domain expert	**Use case:**	CUS011_list_courses
Scenario:	correct	**Term type:**	
Grounding type:		**Ingredient:**	
Language:		**Input:**	
Web method:	listCourses		

e-mail: Info@DynaLearn.eu
website: www.DynaLearn.eu