Ontology Evolution for Experimental Data in Food

Rim Touhami, Patrice Buche, Juliette Dibie, Liliana Ibanescu

Ontology Evolution for Experimental Data in Food

Rim Touhami[1,2], Patrice Buche[2], Juliette Dibie[1], and Liliana Ibanescu[1]

[1] INRA & AgroParisTech, 16, rue Claude Bernard, 75231 Paris Cedex 5, France
[2] INRA & LIRMM, 2, Place Viala, 34060 Montpellier, France
rim.touhami@gmail.com,buche@supagro.inra.fr,
{liliana.ibanescu,juliette.dibie}@agroparistech.fr

Abstract. Throughout its life cycle, an ontology may change in order to adapt to domain changes or to new usages. This paper presents an ontology evolution activity [1] applied to an ontology dedicated to the annotation of experimental data in food [2], and a plug-in, DynarOnto, which assists ontology engineers for carrying out the ontology changes. Our evolution method is an *a priori* method which takes as input an ontology in a consistent state, implements the changes selected to be applied and manages all the consequences of those changes by producing an ontology in a consistent state.

Keywords: ontology evolution, web semantic language

1 Introduction

Ontologies are one of the fundamental layer of the Semantic Web and are designed to represent the knowledge from a domain in terms of concepts (or classes), relations between these concepts and instances of these concepts [3]. An ontology, defined as a formal, explicit specification of a shared conceptualisation [4] may change whenever the domain changes or when domain experts need to add or to restructure the knowledge. When an ontology is used as a system component (the knowledge backbone) of an advanced information system its evolution is a complex process and raise many challenges as, for example: the formal representation of ontology changes, the verification of ontology consistency after applying the ontology changes, and the propagation of those changes to the ontology related entities (e.g. underlying data sets) (see [5] for a complete and detailed overview of the current research activities in ontology evolution).

In [6] we presented a complete system, called ONDINE (ONtology-based Data INtEgration), designed to extract experimental data from tables and store them into a data warehouse with the purpose of enriching incomplete local data sources and to allow afterword a flexible querying of the knowledge base. The backbone of ONDINE system is an ontology which was first designed for predictive microbiology in food [7]. Later a new version of the ontology was designed for the assessment of chemical risk in food. Experimental results of ONDINE system in those two domains were presented in [6]. When ONDINE system was

used for the MAP'OPT project in the domain of food packaging, its ontology had to be adapted to this new domain, and we defined in [2] an Ontological and Terminological Resource (OTR), called naRyQ, dedicated to represent an experiment involving a studied object, some control parameters and a result. During this project, ontology engineers had to manage naRyQ changes. Based on the methodological guidelines given in [1], we propose the workflow given in Figure 1 for carrying out the ontology evolution activity, then we combined, adapted and extended existing approaches in ontology evolution [8–13], and finally we implemented a plug-in, called DynarOnto, to fit naRyQ evolution needs.

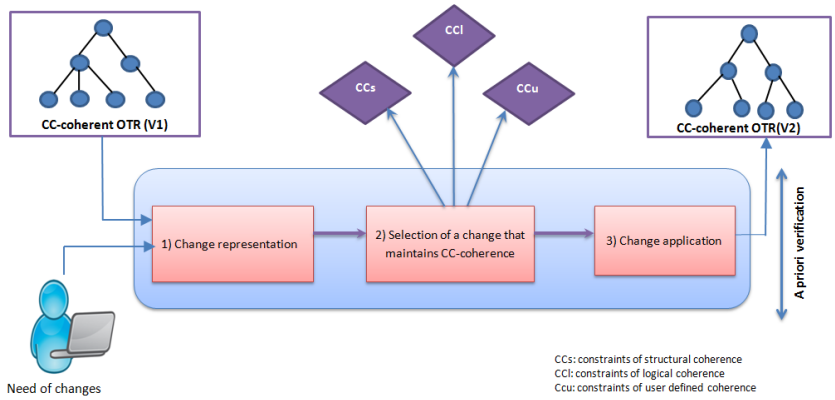

Fig. 1. The ontology evolution activity

Our paper is organized as follows. We first briefly recall the structure of the OTR dedicated to the annotation of experimental data in food and present our definition of its coherence in Section 2. In Section 3, the evolution process is detailed, and we present in Section 4 the implementation and evaluation of our plug-in designed to assist ontology engineering in the evolution activity.

2 naRyQ model and its coherence

In this section, we first recall naRyQ model presented in [2] then we define the coherence constraints it must respect.

2.1 naRyQ model

naRyQ (*n-ary Relations between Quantitative experimental data*) is designed to represent experiments in order to annotate data tables representing scientific experiments results in a given domain (see [2] for more details). As recommended by W3C [14] experiments which involve a studied object, several experimental parameters and a result are represented using n-ary relations without distinguished

arguments. More precisely "pattern 1" is used and it consists in representing a n-ary relation thanks to a concept associated with its arguments via properties.

Example 1. Let us consider the experiment where the permeability, which is the experiment's result, of a given packaging, which is the studied object, is studied in a set of control parameters specified by the packaging thickness, the temperature and the differential partial pressure. This experiment with 6 arguments can be represented by a n-ary relation *Permeability_Relation* as given in Figure 2.

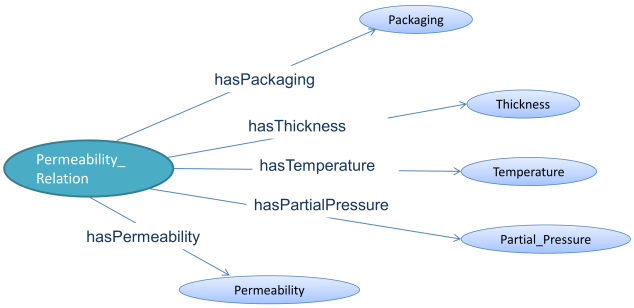

Fig. 2. n-ary relation Permeability_Relation.

The conceptual component of naRyQ is composed of a *core ontology* to represent n-ary relations between experimental data and a *domain ontology* to represent specific concepts of a given application domain. Figure 3 gives an excerpt of naRyQ in the domain of risk in food packaging. The representation of n-ary relations between experimental data requires a particular focus on the management of quantities. In the up core ontology, generic concepts *Relation_Concept* and *Argument* represent respectively n-ary relations and arguments. In the down core ontology, generic concepts *Dimension*, *UM_Concept*, *Unit_Concept* and *Quantity* allows the management of quantities and their associated units of measure. The sub-concepts of the generic concept *Symbolic_Concept* represent the non numerical arguments of n-ary relations between experimental data. The **domain ontology** contains specific concepts of a given application domain. They appear in naRyQ as sub concepts of the generic concepts of the core ontology. All concepts are represented as OWL classes, hierarchically organized by the subsumption relation *subClassOf* and pairwise disjoints.

The terminological component of naRyQ contains the set of terms describing the studied domain and are used to annotate data tables. Sub concepts of the generic concepts *Relation_Concept*, *Symbolic_Concept* and *Quantity*, as well as instances of the generic concept *Unit_Concept*, are all denoted by at least one term of the terminological component. Each of those sub concepts or instances are, in a given language, denoted by a preferred label and optionally by a set of alternative labels, which correspond to synonyms or abbreviations. Labels are

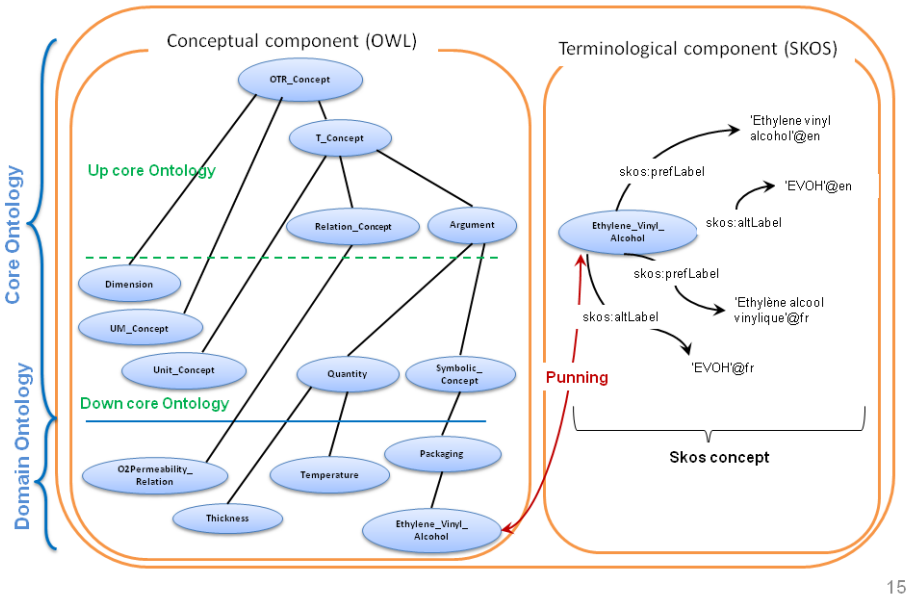

Fig. 3. An excerpt of naRyQ in the domain of risk in food packaging.

associated with a concept or an instance thanks to SKOS labeling properties[3] recommended by W3C. For instance, in Figure 3, English terms *Ethylene vinyl alcohol* and *EVOH* denote the symbolic concept *Ethylene_Vinyl_Alcohol*.

2.2 naRyQ $\mathcal{CC}$-coherence

In [11] the coherence of an ontology is classified in three categories: i) structural coherence which is related to constraints of the ontology's representation languages, ii) logical coherence where the semantic correctness of the ontology's entities are checked and iii) user-defined coherence which refers to specific user requirements and constraints related to the ontology's context of use. Inspired by [8] we define a set of conditions that the ontology must respect for each category of coherence. These conditions are called Coherence Constraints, denoted by $\mathcal{CC}$-constraints. An ontology is $\mathcal{CC}$-coherent if it respects a set of defined $\mathcal{CC}$-constraints. We briefly present below the $\mathcal{CC}$-constraints for naRyQ.

Structural $\mathcal{CC}$-coherence We have modeled naRyQ using a subset of OWL2-DL entities (i.e. classes, properties, constructors of classes, etc.) and axioms (see the technical report [15] for more details).

[3] http://www.w3.org/TR/skos-reference/

To define structural coherence associated to the conceptual part of naRyQ, the constraint defined by the W3C group, which says that every OWL axiom must be well defined[4] was extended to associate it with each axiom and constructor used in the modeling of naRyQ: if an entity refers to an other entity, this latest must be defined in the OTR. We defined 15 structural $\mathcal{CC}$-constraints, denoted by $\mathcal{CC}$s. We present below two examples of $\mathcal{CC}$s.

- CS1- Each anonymous class defined by a value restriction, `owl:allValuesFrom` or `owl:someValueFrom` links a pair property-class or property-data type. The property and the class used in the definition of the anonymous class must be defined in the OTR.
- CS2- Each instance of `skos:concept` must have at least a preferred label.

Logical $\mathcal{CC}$-coherence We defined 6 logical $\mathcal{CC}$-constraints, denoted by $\mathcal{CC}$l, which take into account the subset of OWL constructors and axioms used to model naRyQ. Those constraints are taken from the literature [12]. We present below two examples of $\mathcal{CC}$l.

- CL1- A class can not be disjoint with its superclass.
- CL2- If two values restrictions `owl:allValuesFrom` which connect a pair property-concept are associated with the same concept c, then the concepts defining the restrictions cannot be disjoint.

User-defined $\mathcal{CC}$-coherence The user-defined $\mathcal{CC}$-constraints, denoted by $\mathcal{CC}$u, correspond either to quality criteria modeling [8] or to specific criteria corresponding to the modeled task. We defined 9 generic $\mathcal{CC}$u which refer to quality criteria modeling and 20 new $\mathcal{CC}$u which are specific to the annotation task, detailed in [15]. We present below two examples of $\mathcal{CC}$u.

- CU1- A n-ary relation has at least two arguments.
- CU2- Each quantity must be associated with its units of measurement (at least one) through the `owl:allValuesFrom` restriction and the property `hasUnitConcept`. The units of measurement are defined by enumeration using the `owl:oneOf` constructor.

3 Coherent evolution of naRyQ

As suggested in [1] our ontology evolution activity consists in applying changes to an ontology while preserving its $\mathcal{CC}$-coherence, i.e. respecting all its $\mathcal{CC}$-constraints presented in the previous section (i.e. structural, logical and user-defined constraints). Figure 1 presents the evolution process composed of three main steps. The first step consists in presenting all the possible changes for naRyQ evolution to the ontology engineer. From this list of changes, the ontology

[4] `http://www.w3.org/TR/owl-ref/#OWLDL`

engineer chooses the ones to be applied. The second step consists in preserving the $\mathcal{CC}$-coherence of naRyQ during its evolution. To do this, an additional set of changes is added automatically to the requested changes in order to maintain *a priori* (i.e. before the application of the requested changes) the $\mathcal{CC}$-coherence of the OTR. In the third and last step, requested and additional changes are applied to the OTR. In the following, we present the two first steps.

3.1 Change representation

This first step consist in presenting to the ontology engineer all the possible changes for naRyQ. In order to generate all possible changes we first identified all entities and axioms used to model naRyQ in OWL2-DL and SKOS. We selected 55 changes from the literature and defined 26 new changes to take into account the specificity of naRyQ. The complete list is available in [15]. Among these changes, some changes are not accessible to ontology engineer but are rather used in the kits of changes, presented in the following section. Table 1 contains a subset of changes required for the evolution of naRyQ, where changes in bold are new ones compared to the literature.

Change	**Element**	**Role**
addClass	NamedClass	add an OWL class to an ontology
addSubClassOf	SubClass	add a subsumption link
updateDomain	Domain	update the domain of a relation
updateSomeValuesFromRestriction	SomeValuesFrom	update an existential restriction
addDataTypeRestriction	DataTypeRestriction	add a value range
deleteFromAllDisjointClasse	DisjointClasses	delete a class of a set of disjoint classes
updatePrefLabel	SkosPrefLabelAssertion	update a preferred label of a SKOS concept

Table 1. A subset of change operations for naRyQ

3.2 Selection of a change that maintains $\mathcal{CC}$-coherence

After applying a change, one or more $\mathcal{CC}$-constraint may be violated (e.g. adding a new concept which represents a n-ary relation violates CU1 constraint presented in Section 2.2). Hence, a second step is necessary to restore the $\mathcal{CC}$-coherence of the OTR while applying a change. To achieve this goal, we adapted the notions of kit of changes [9] and of strategy [8] to our needs.

Kit of changes To maintain a priori the $\mathcal{CC}$-coherence of naRyQ during its evolution, we associated a kit of changes with each change that violates one or more $\mathcal{CC}$-constraints and wich is accessible to ontology engineer. Its definition takes into account all the $\mathcal{CC}$-constraints which can be violated by the requested changes.

Definition 1. Given a $\mathcal{CC}$-coherent ontology O and a change c. If the application of c to O doesn't maintain the $\mathcal{CC}$-coherence of O, then **a kit of changes** is associated with c. It is composed of:

- preconditions: a set of assertions which must be true in order to apply c to O;
- the change c;
- mandatory additional changes: a set of changes which are attached to c in order to correct the inconsistencies that may occur in O when c is applied to it;
- optional additional changes: a set of changes which can be applied in addition to mandatory additional changes.
- post-conditions: a set of assertions which must be true after the application of c to O.

Two examples of kits of changes are presented in [15] and 15 kits from 63 were implemented.

Definition 2. The *addClass* change allows a new class to be added to the OTR. The kit of changes associated with the *addClass* change is defined in the following: Because of the constraint *"An ontology should not contain two concepts with the same URI"*, the **precondition** states that a new class *newc* can be added to the set of concepts C of naRyQ if its URI doesn't already exist in C. The requested change, which is the creation of the new class *newc*, is then applied to naRyQ. Unfortunatly, the application of this change violates several $\mathcal{CC}$-constraints of the OTR. The application of a set of **mandatory additional changes** is then required:

1. The additional changes `addSkosConcept` and `addPrefLabel` are triggered to resolve the $\mathcal{CC}$-incoherence linked to the terminological part of naRyQ due to the constraint *"Each n-ary relation and each argument is associated with a terminological part"*.
2. The creation of a disjunction relationship between the new class and all its sibling classes resolves a second $\mathcal{CC}$-incoherence due to the contraint which requires that *"n-ary relations and their arguments should be mutually disjoint"*.
3. If the new class *newc* is a n-ary relation, then at least two arguments must be associated with *newc*. Additional changes `addAllValuesFromRestrictionToClass` or `addSomeValuesFromRestrictionToClass` or `addHasValueRestrictionToClass` are applied.
4. If the new class *newc* is a n-ary relation, then exact (or greater than or equal to 1) cardinality restrictions must be associated with the mandatory arguments of *newc* to ensure their existence at the instance level.
5. If the new class *newc* is a quantity, then a dimension and a unit of measurement must be associated with it.
6. Adding a specialization relationship between *newc* and its superclass c resolves the $\mathcal{CC}$-incoherence due to the constraint *"Each concept must be connected by a relation of specialization to at least one other named concept.*

Besides these mandatory additional changes, it is possible to apply other optional additional changes which are not necessary to maintain the $\mathcal{CC}$-coherence of naRyQ but which may be requested by the ontology engineer. These optional additional changes may be for instance: adding other prefered and alternative labels to the new class, adding other arguments to the new class if it is a n-ary relation or other units of measurement if it is a quantity.

It is important to notice that each mandatory or optional additional change can in turn cause other $\mathcal{CC}$-incoherences which can be resolved by triggering the required kits of changes. So each additional change can also call a kit of changes.

Evolution strategies The kit of changes defined in Definition 2 can be used when there is only one possible solution to solve an incoherence. But sometimes there can be several possible solutions to correct violated constraints. In this case, we have to use evolution strategies [8] in order to represent the different alternatives.

Definition 3. Given a $\mathcal{CC}$-coherent ontology O and a change c, if the application of c to O leads to several possible alternatives to maintain the $\mathcal{CC}$-coherence of O, then we call **evolution strategy** the choice made between these alternatives.

There exist two types of kits of changes: those with evolution strategies and those without. The kit of changes associated with the change *addClass*, presented above, is an example of a kit without evolution strategies. Kits with evolution strategies can be considered as a variant of the first type since they need to fix the strategy (i.e. the choice) before applying the kit.

Example 2. In order to delete a concept, there are several solutions to deal with its sub concepts and there are also several solutions to deal with its terms:

- To deal with orphaned concepts (sub concepts of deleted concept), the possible solutions are i) delete them, 2) reconnect them to super concepts or 3) reconnect them to the root.
- To deal with terms of deleted concept, the possible solutions are: 1) delete them or 2) reconnect them to super concepts.

4 Implementation and evaluation

4.1 Implementation

We implemented the ontology evolution activity presented above as a Protégé plug-in called DynarOnto. Figure 4 shows the different designed menus:

- The *Relation Changes* menu presents the changes that can be applied to a n-ary relation (e.g. add a n-ary relation, delete a n-ary relation, update the arguments of a n-ary relation, etc.).
- The *Argument Changes* menu presents the changes that can be applied on arguments of a n-ary relation (i.e. Quantity and Symbolic_Concept).

– *Evolution Parameters* menu helps the ontology engineer to define its evolution strategy that will be taken into account during the OTR evolution.

The screen shot presented in Figure 4 is the interface for the kit of changes associated with the change *addClass* presented in Definition 2. Let us consider that we want to add the new n-ary relation *CO2Permeability_Relation*, which is a sub relation of the existing n-ary relation *Permeability_Relation*. *CO2Permeability_Relation* is defined by the following arguments: Packaging, Thickness, Temperature, Partial_pressure_difference, **Relative_humidity** and **CO2_permeability**. Arguments presented in bold are specific to the sub n-ary relations. The other arguments are inherited from its super n-ary relation *Permeability_Relation* (see Figure 2). To add this new n-ary relation, ontology engineer starts by clicking on *Add relation* of the *Relation changes* menu. Using the panel of the displayed interface, the ontology engineer enters the name of the new n-ary relation, *CO2_permeability_Relation*. Then, he chooses *Permeability_Relation* as his parent class in the hierarchy of n-ay relations of the OTR displayed by clicking on "Choose" button. By confirming the choose of the n-ary relation's parent, inherited arguments appears in panel 3 of the displayed interface. The ontology engineer can both specialize inherited arguments and add new ones by indicating their numbers in panel 2. To define the new n-ary relation *CO2_permeability_Relation*, ontology engineer should add the new argument *Relative_humidity* and specialize the inherited argument *Permeability* in *CO2_permeability*, using the panel 2. To associate terminology to the new n-ary relation, ontology engineer uses panel 4. Finally, when the ontology engineer clicks on "OK" button to validate the add of *CO2_permeability_Relation*, DynarOnto plugin checks that the associated preconditions are verified (e.g. *CO2_permeability_Relation* does not exist in the OTR). The interface facilitates the verification of some preconditions. For instance, when the ontology engineer decides to specialize an argument, the plugin displays only the hierarchy of more specific concepts. Once the preconditions are checked, the plugin applies the requested change and the set of additional mandatory changes, allowing violated $\mathcal{CC}$-constraints to be resolved.

4.2 Evaluation of DynArOnto plugin

DynArOnto was evaluated in an incremental way by ten users, with different backgrounds.Three evaluation sessions were organized, DynArOnto interface being improved after each session. Users were asked to apply three changes (add, update n-ary relations and delete arguments). A questionnaire was created to collect their evaluations. Most participants affirm that using DynArOnto to manage the evolution is easier than using Protégé. It is also reflected in the time spent by users to apply the changes: the add relation change (resp. update/suppress change) was done with Protégé in an average of 42 minutes (resp. 11 minutes). Using DynArOnto, the duration was reduced by 20% (resp. 27%). Finally, DynArOnto helped the users to manage the evolution of an OTR dedicated to the annotation of experimental data while guaranteeing its $\mathcal{CC}$-coherence, which was

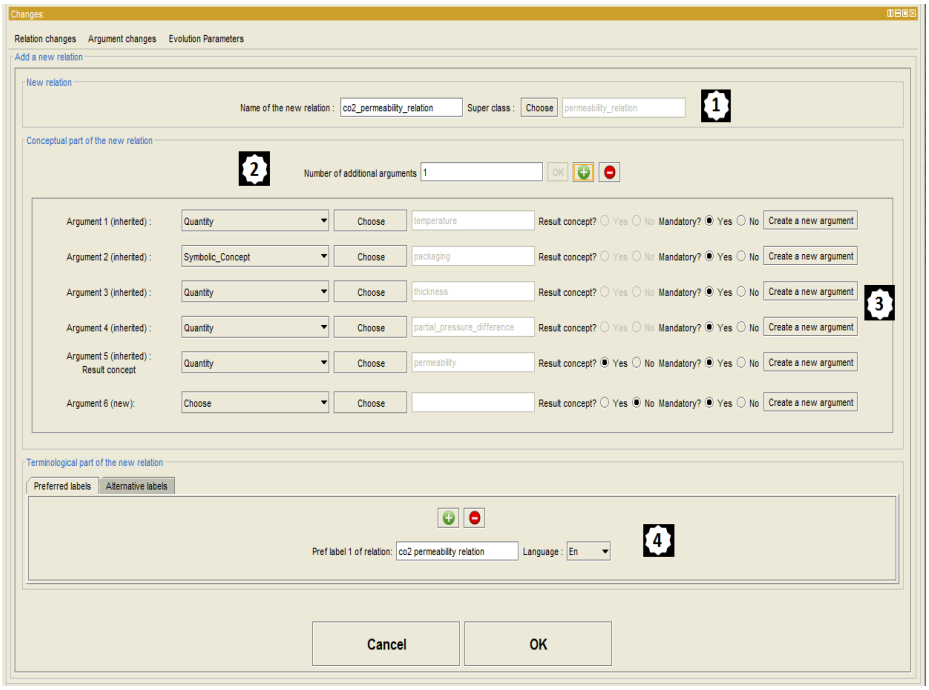

Fig. 4. Screen shot of the interface to add a new n-ary relation

not the case in Protégé where the users were not guided and make some errors (e.g. incorrect use of OWL restrictions used to link arguments to n-ary relations, forgetting to associate terms to concepts, etc.).

5 Related work

In [5], a recent overview of the current research activities in ontology evolution, authors give first the different definitions of ontology evolution, then they present and discuss the various process models that were proposed for the ontology evolution tasks and afterword they propose a unified ontology evolution cycle thus providing a unique overview over several research fields. Our evolution method is defined as in the NeOn Glossary of ontology engineering tasks [1] which states that ontology evolution is "the activity of facilitating the modification of an ontology by preserving its consistency" and this is a narrower view than in [5].

To maintain the coherence of an ontology during its evolution, the authors of [11–13] have proposed *a posteriori* approaches. This type of approach allows the application of changes in order to evaluate the evolution impact on the ontology, then suggests how to repair inconsistencies in case of problems. To avoid backtracking after modification, resulting in a loss of time and resources, *a priori* approaches, where the coherence checking is made before the application

of changes, have been proposed in the literature [9, 10, 16]. The work of Stojanovic [8] is the first to propose an ontology evolution process defined for KAON ontologies and using strategies for the task of managing changes. In [9] authors define kits of changes in order to manage the inconsistencies generated by each change. The authors of [10] proposes a system of evolution of an OTR dedicated to the semantic annotation of text documents. Nevertheless their approach maintains the coherence of the OTR and the semantic annotations without defining explicitly the type of coherence which is considered. In [9], a preventive approach that manages the inconsistencies generated by each change is described. A set of rules that must be maintained during the evolution of an ontology is defined but, as in [10], the authors don't define explicitly the type of coherence which is considered. More recently, the authors of [16] proposed a framework based on graph rewriting rules that maintains a set of constraints. However, their approach takes into account only one way to manage the inconsistencies after applying changes (e.g. the remove of a class results automatically in removing all its instances). To the best of our knowledge, our preventive approach is the first one which is based on a clear definition of ontology coherence and simultaneously permits to manage evolution of an ontology involving quantitative data.

6 Conclusion

We have proposed in this paper an ontology evolution activity for an OTR dedicated to the annotation of experimental data which preserves its $\mathcal{CC}$-coherence. To do this, we first identified all the necessary constraints (i.e $\mathcal{CC}$s, $\mathcal{CC}$l and $\mathcal{CC}$u) to be checked in order to avoid possible inconsistencies. Secondly, we identified all the required changes for the evolution of naRyQ. Then, we identified all the $\mathcal{CC}$-incoherences that can occur after the application of each change. In order to solve these $\mathcal{CC}$-incoherences, we defined a kit of changes for each change that violates one or more $\mathcal{CC}$s-constraints. A kit of changes allows the $\mathcal{CC}$-coherence of naRyQ to be preserved *a priori* by checking a set of preconditions, by applying a set of additional changes and/or by using evolution strategies. The originality of our method is to propose an evolution of an OTR containing inter dependent concepts to manage quantitative data. Further work is to explore how to propagate changes to all the ontology related artifacts: individuals, mapping, applications, metadata.

Acknowledgements

This work has been supported by the French Research Agency (ANR) in the framework of ALIA/MAP'OPT project, by the Carnot Institute 3BCAR in the framework of the IC2ACV project and by the Labex NUMEV.

References

1. Palma, R., Zablith, F., Haase, P., Corcho, Ó.: Ontology evolution. In del Carmen Suárez-Figueroa, M., Gómez-Pérez, A., Motta, E., Gangemi, A., eds.: Ontology Engineering in a Networked World. Springer (2012) 235–255
2. Touhami, R., Buche, P., Dibie-Barthélemy, J., Ibănescu, L.: An Ontological and Terminological Resource for n-ary Relation Annotation in Web Data Tables. In: OTM 2011 (2). Volume 7045 of LNCS., Springer (2011) 662–679
3. Guarino, N., Oberle, D., Staab, S.: What is an ontology? In Staab, S., Studer, R., eds.: Handbook on Ontologies. International Handbooks on Information Systems. Springer Berlin Heidelberg (2009) 1–17
4. Studer, R., Benjamins, V.R., Fensel, D.: Knowledge engineering: Principles and methods. Data and Knowledge Engineering **25**(1-2) (March 1998) 161–197
5. Zablith, F., Antoniou, G., d'Aquin, M., Flouris, G., Kondylakis, H., Motta, E., Plexousakis, D., Sabou, M.: Ontology evolution: a process-centric survey. Knowledge Eng. Review **30**(1) (2015) 45–75
6. Buche, P., Dibie-Barthélemy, J., Ibanescu, L., Soler, L.: Fuzzy web data tables integration guided by an ontological and terminological resource. IEEE Trans. Knowl. Data Eng. **25**(4) (2013) 805–819
7. Buche, P., Couvert, O., Dibie-Barthélemy, J., Hignette, G., Mettler, E., Soler, L.: Flexible querying of web data to simulate bacterial growth in food. Food Microbiology **28**(4) (2011) 685–693
8. Stojanovic, L.: Methods and Tools for Ontology Evolution. PhD thesis, University of Karlsruhe, Germany (2004)
9. Jaziri, W., Sassi, N., Gargouri, F.: Approach and tool to evolve ontology and maintain its coherence. Int. J. Metadata Semant. Ontologies **5**(2) (2010) 151–166
10. Tissaoui, A., Aussenac-Gilles, N., Hernandez, N., Laublet, P.: EvOnto - joint evolution of ontologies and semantic annotations. In: KEOD 2011. (2011) 226–231
11. Haase, P., Stojanovic, L.: Consistent evolution of owl ontologies, Springer (2005) 182–197
12. Djedidi, R.: Approche d'évolution d'ontologie guidée par des patrons de gestion de changement. Thèse de doctorat, Université Paris-Sud XI (Nov 2009)
13. Khattak, A.M., Latif, K., Lee, S.: Change management in evolving web ontologies. Knowledge-Based Systems **37**(0) (2013) 1 – 18
14. Noy, N., Rector, A., Hayes, P., Welty, C.: Defining n-ary relations on the semantic web. W3C working group note http://www.w3.org/TR/swbp-n-aryRelations.
15. Touhami, R.: Definition of naRyQ OTR CC-Constraints and kits of changes. Technical report, INRA (April 2015) http://umr-iate.cirad.fr/content/download/4838/36843/version/1/file/CC-coherence.pdf.
16. Mahfoudh, M., Forestier, G., Thiry, L., Hassenforder, M.: Algebraic graph transformations for formalizing ontology changes and evolving ontologies. Knowl.-Based Syst. **73** (2015) 212–226