Closing Ontologies to Define OLAP Systems

Manuel Torres[1], José Samos[2], Eladio Garví[2],

[1]University of Almeria, Almeria, Spain

[2] University of Granada, Granada, Spain

ABSTRACT

Ontologies can be used in the construction of OLAP (On-Line Analytical Processing) systems. In such a context, ontologies are mainly used either to enrich cube dimensions or to define ontology based-dimensions. On the one hand, if dimensions are enriched using large ontologies, like WordNet, details that are beyond the scope of the dimension may be added to it. Even, dimensions may be obscured because of the massive incorpora- tion of related attributes. On the other hand, if ontologies are used to define a dimension, it is possible that a simplified version of the ontology is needed to define the dimension, especially when the used ontology is too complex for the dimension that is being defined. These problems may be solved using one of the existing mechanisms to define ontology views. Therefore, concepts that are not needed for the domain ontology are kept out of the view. However, this view must be closed so that, no ontology component has references to components that are not included in the view. In this work, two basic approaches are proposed: enlargement and reduction closure.

Keywords: OLAP, Ontologies, OWL, Tourism, Views

1. INTRODUCTION

Ontologies are applied in a wide range of application fields such as knowledge management, natural language processing, e-commerce, intelligent information integration, information retrieval, Semantic Web, and so on. Another emerging application area is data warehousing, in particular, in the construction of Multidimensional information systems, also known as OLAP (On-Line Analytical Processing) systems (Thomsen, 2002). In such a context, ontologies may be used for several purposes, as to address semantic heterogeneity (Priebe & Pernul, 2003; Niemi et al., 2007). But in this paper, the two applications of ontologies in OLAP systems we are interested in are the use of ontologies to enrich cube dimensions (Mazón, Trujillo, Serrano & Piattini, 2006; Mazón & Trujillo, 2006; Pardillo & Mazón, 2011; Neumayr, Schütz, & Schrefl, 2013), and the definition of ontology based-dimensions (Romero & Abelló, 2007a, 2007b, 2010).

With respect to the first use, dimension enrichment may be carried out obtaining from existing ontologies new relationships (adding new aggregation levels), or obtaining another semantic relationships (e.g. the *whole-part* relationship), so that new hierarchies may be added to the dimensions. In (Mazón, Trujillo, Serrano & Piattini, 2006; Mazón & Trujillo, 2006), the use of WordNet (Miller et al, 1990) for enriching data warehouse dimensions is proposed, based on the semantic relationships defined between WordNet concepts. Therefore, if a *Product* dimension is used in our OLAP system, using the semantic relationships defined on WordNet, the *subtype, type*, and *class* of a product can be obtained. (For example, if we have a product named as Chardonnay, we can obtain that it is White wine, Wine, and an Alcoholic beverage.) But, if dimensions are enriched with every semantic relationship available in WordNet, details that are beyond the scope of the dimension may be added to it. Even, dimensions may be obscured because of the massive incorporation of related attributes.

With respect to the other use of ontologies in data warehousing, the definition of dimensions using existing ontologies, it makes easier dealing with integration and interoperability issues when data warehouses share some dimensions. However, it is possible that these ontologies we are using to define the dimensions include concepts that are not needed for the dimensions.

Therefore, on the one hand, dimension enrichment using existing ontologies may lead to obtain dimensions that include attributes that have not to be included in the dimension; on the other hand, the definition of dimensions using ontologies may force that an OLAP tool has dimensions that that incorporate concepts that are not needed. One way or another, these problems may be solved using one of the existing mechanisms to define ontology views (Magkanaraki et al., 2004; Rajugan et al, 2005; Uceda-Sosa et al., 2004; Volz et al., 2003; Noy et al., 2004; Seidenberg & Rector, 2006) Therefore, concepts that are not needed for the domain ontology are kept out of the view. However, this view must be *closed*. The *closure* property is a well-known property of schemas in object databases, so that no class of a schema has references to classes that are not included in the schema (Rundensteiner, 1992; Lacroix & Delobel, 1998; Torres & Samos, 2001). Analogously, an ontology will be closed if every component of the ontology does not have references to components that are not included in the ontology.

In order to achieve the *ontology closure* property, in this work we propose two basic approaches, named as *enlargement closure* and *reduction closure*.

Enlargement closure recursively includes each component that is referenced by ontology components, so that the original ontology is *enlarged* with the referenced components. The main drawback of the enlargement closure is that the systematic inclusion of referenced components in non-closed ontologies may lead, in certain situations, to obscure the ontology that was originally specified by the ontology definer.

In contrast to enlargement closure, reduction closure is based on the assumption that the ontology definer is interested only in the components that he or she has selected, and no one else. If the ontology includes some components with external references, in order to guarantee the ontology closure, but without adding more components, ontology components that have external references are replaced with *views* that hide those references. Once all the components with external references have been replaced with views, the effect of this replacement must be propagated over the ontology (e.g., updating references or generating the needed intermediate views). For such a purpose, a set of rules to be applied and an algorithm to reach ontology closure under the reduction approach have been defined, making easier the definition of ontology views.

The remainder of this paper is organized as follows. Section 2 describes the concepts of enlargement and reduction closure. Section 3 describes how closure concepts can be applied to OWL ontologies. In Section 4, a tool we have developed to apply the proposed closure approaches is briefly described. In Section 5, the related work is described. Finally, we conclude the paper in Section 6.

2. ONTOLOGY CLOSURE

Ontology closure is inspired in a well-known property of databases named as schema closure, which is mainly studied in the context of object databases (Rundensteiner, 1992; Lacroix & Delobel, 1998; Torres & Samos, 2001). In the ontology context, ontology closure states that no component of an ontology includes references to other components that are not included in the ontology. A *closed* ontology can be defined in this way: Let *C* be the set of components of an ontology and *R* the set of inheritance relationships defined in it. $Uses(C_i)$ can be defined as the set of components used by the properties of C_i.

Definition 1 (Closed ontology). An ontology O = (C, R) is closed if, and onlyif,C=Uses(Ci)∪C,∀Ci ∈C.

To reach ontology closure, in this paper two approaches are proposed: enlargement closure and reduction closure. These approaches are studied in the next subsections, but before dealing with them, an example is introduced. This example will be used along the paper to illustrate each of the proposed approaches. The example is an excerpt of a tourism ontology taken from (Knublauch, 2014). In our ontology, destinations (travels) are modelled. Destinations have activities, and these activities have a contact person. Destinations are specialized in beaches, rural areas and urban areas. Activities are specialized in sports, relaxation and adventures. Finally, destinations include recommendations with their respective ratings. Figure 1 illustrates the tourism ontology.

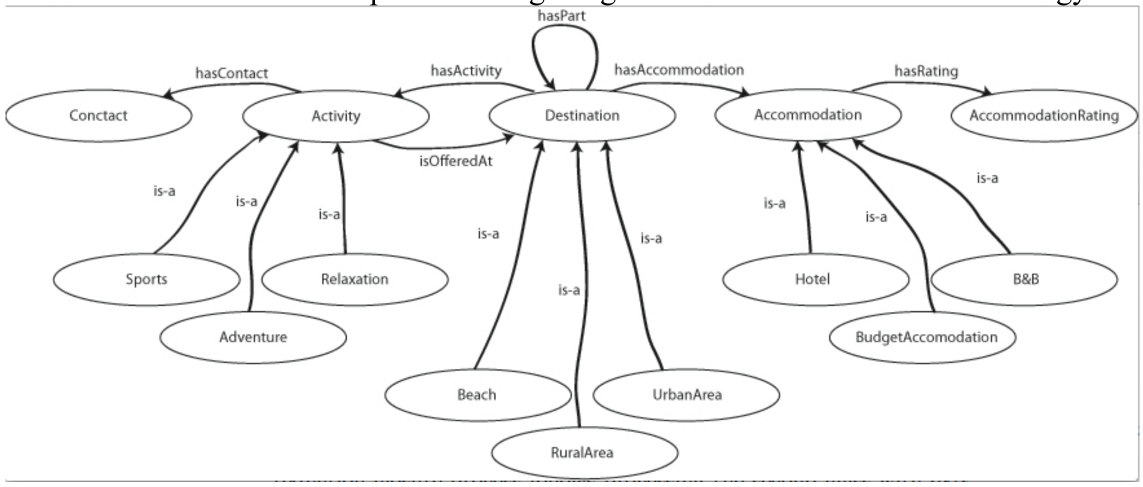


Figure 1: Tourism ontology

Once the example has been introduced, the proposed closure approaches are described.

2.1. ENLARGEMENT CLOSURE

Enlargement closure is based on an algorithm proposed in (Rundensteiner, 1992) to include in a view schema of an object database every class referenced by a class of the view schema. In the ontology context, each component (e.g., classes or properties) referenced by another component of the ontology must also be included in the ontology.

This kind of closure is based on the idea that the user wants an ontology with the components that he or she has selected and, if it is necessary, other components can also be included into it in order to achieve the ontology closure. Below, the enlargement closure algorithm is shown.

```
Function EnlargementClosure (O): NeededComponents
1. Temp = C = GetComponents(O); NeededComponents = $\emptyset$
2. while Temp $\neq$ $\emptyset$
3.   $C_i$ = GetAndRemoveNext(Temp)
4.   if $C_i \in$ C and $C_i \in$ NeededComponents then
```

```
5.   NeededComponents = NeededComponents ∪ $C_i$
6.   end if
7.   for all $C_k \in Uses(C_i)$
8.   if $C_k \in$ C and $C_k \in$ Temp and $C_k \in$ NeededComponents then
9.   Temp = Temp ∪ $C_k$
10.   end if
11.   end for
12. end while
13. return NeededComponents
```

This algorithm takes as input the set of components `O` selected to make up the view ontology. The algorithm returns in `NeededComponents` the components that are needed to close the ontology.

The ontology `O` only consists of the set of ontology components `C` specified by the ontology definer. This set is obtained by the function `GetComponents` that is not described here. The set of components `C` that make up the ontology `O` is copied to an auxiliary variable `Temp` which will be used to check the ontology closure.

The algorithm processes the components of `Temp` one by one pulling out from this set by a function `GetAndRemoveNext`, that is not described here. If the component that is being processed does not belong to the set of ontology components or to the set of needed components (i.e. `NeededComponents`), it is added to the set of needed components. Checking whether `NeededComponents` is empty we can know if the ontology is closed.

Next, the components used by the current component are added to `Temp` if they have not been already added; this is checked at this way: a component does not have to be added to `Temp` if it is a component of `O` (it was already included in `Temp` at the beginning of the algorithm), if it is a component that is already in `Temp` (it is a component that is not still processed) or if it belongs to `NeededComponents` (it is an external reference added by another component).

2.2. REDUCTION CLOSURE

Enlargement closure may add some components to view ontologies because all the referenced components are also included into the ontology, as well as all the components referenced by them, and so on. However, under certain situations the ontology definer may not want to include into the view ontology all the referenced components, or some of them. However, since ontology closure must be fulfilled, but no additional components are wanted, components with external references must be replaced with another components, which do no not include the external references. This is the premise which reduction closure is based on, that is, to replace components that have external references in order to remove such references. Thus, ontology closure is also fulfilled and no components are added to the ontology.

Summarizing, reduction closure assumes that the user wants to include only the components that he or she has selected and no one else. In order to remove the references to non-included components, new view classes have to be defined to replace them. Components with external references cannot be directly modified because, if we modify them instead of defining new ones, we would produce collateral effects in other ontologies where those components are included or used (e.g. other view ontologies).

Reduction closure is carried out by an algorithm that takes as input the set of components `S` selected to make up the ontology. If the ontology is not closed, the algorithm returns the set `S` incorporating the needed modifications, so that the closure of the ontology is reached.

```
Procedure ReductionClosure (S)
1. List = FindComponentsWithExternalReferences (S)
2. if List ≠ NIL
3.  List = PropagateChanges (S, List)
4. UpdateOntology (S, List)
5. end if
```

The algorithm works as follows. First, the function `FindComponentsWith-ExternalReferences` is called to obtain the set of components that have external references (step 1). If the ontology is closed, this function returns an empty list. However, if the ontology is not closed, this function returns the list of nodes corresponding to classes with external references.

Once the existence of external references has been checked, if the ontology is not closed, then a non-empty list is returned and the function `PropagateChanges` is called (steps 2 and 3). If there exists any class affected by the replacement of classes having external references to view classes, this function includes new nodes to the list. The function updates the list propagating the modifications following the rules depicted in Figure 2, generating a new node for each affected class, and updating existing nodes. After propagating the changes, the list contains a node for each class that must be replaced with a new view class, so that the ontology does not have external references. All the needed information to define the new components is stored in the nodes.

Finally, when the list is built, new views must be defined. Then, the ontology is updated replacing with new view classes those classes that have external references as well as the affected ones. This process is carried out by the function `UpdateOntology` (step 4), which follows the rules described in Figure 2.

In order to obtain the classes that must be replaced, we propose a set of rules to decide whether a component has to be replaced with a view component to update its references or not. If these rules are applied, the set of components to be modified is obtained.

Obtaining the classes to be modified

The rules described below show the set of components that are affected by the replacement of classes having external references. These rules consider the five cases that may occur, which are illustrated in Figure 2. In the figure, modified classes are marked with an asterisk (*), and the dotted arrow points to affected classes.

- If `A` is modified and `B` is referenced by `A`, `B` does not need to be modified because of the modification of `A` (Figure 2.a).
- If `A` has a reference to `B`, and `B` is modified, `A` must be modified (Figure 2.b).
- If `A` and `B` are mutually referenced, and at least, one of them has to be modified, both components have to be adapted (Figure 2.c).
- If `A` must be modified and `B` specializes `A`, `B` must be modified after modifying `A` (Figure 2.d).
- If `A` generalizes `B`, and `B` must be modified, `A` does not need to be modified because of the modification of `B` (Figure 2.e).

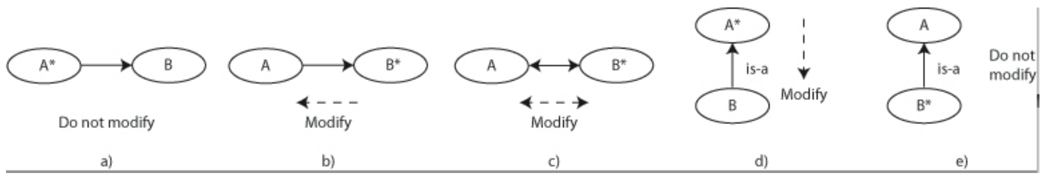


Figure 2: Cases for propagating modifications of components of an ontology (Components marked with an asterisk represent modified components)

Following these indications, a list of nodes can be built where nodes symbolize classes to be modified, either because of having external references, or to propagate the modifications. The node structure is depicted in Figure 3.a. Each node contains the name of the class to be modified (`OldClassName`) and the name of the class that is going to replace it (`NewClassName` -this name may be generated automatically adding a numerical suffix indicating the number of classes defined from its base class). In addition, the node includes the list of classes included in the ontology that are referenced by the class (`ReferencesTo`), the list of external references that must be deleted (`ToBeDropped`), and the list of references that must be updated (`ToBeUpdated`). Adding the list of external references (`ToBeDropped`) in each node makes easier the later definition of the class that will replace the class corresponding to that node in order to hide the external references. The list `ReferencesTo` is used to check whether a class has references to classes that have been replaced with new classes, and therefore, must update its references.

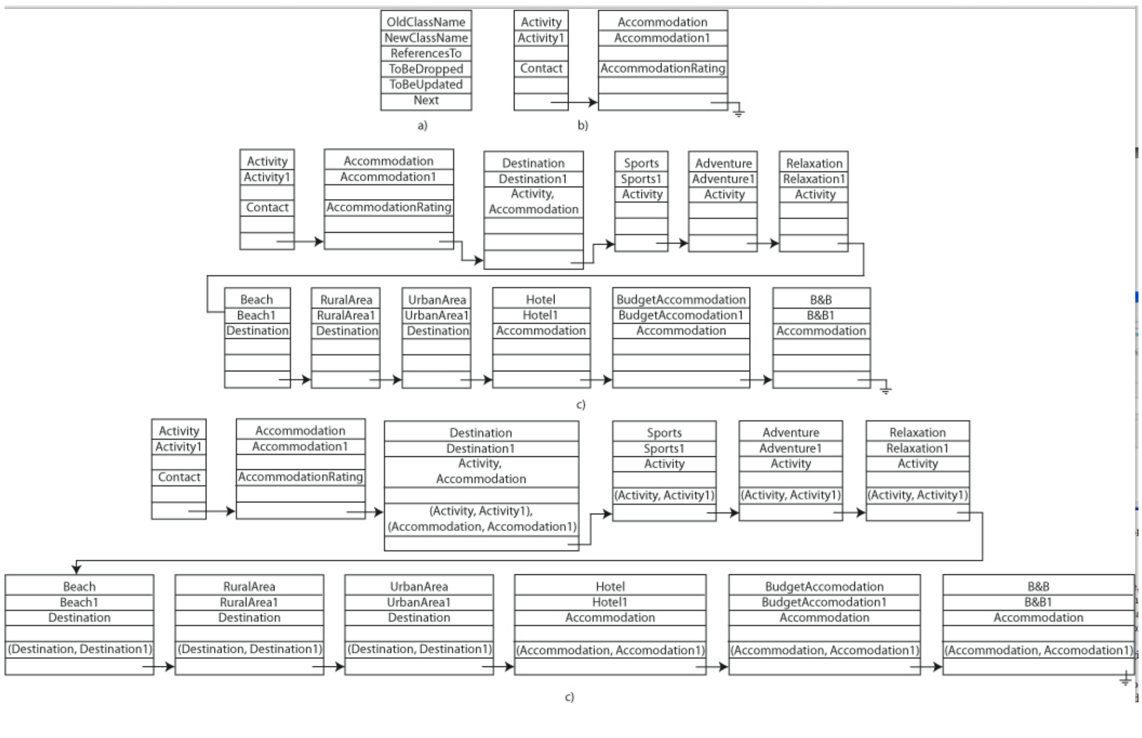


Figure 3: a) Node structure corresponding to classes to be modified; b) Node list corresponding to classes with external references; c) Resulting node list after applying the five rules; d) Final node list, where references to new classes have been updated to make easier the definition of view classes

3. CLOSING OWL ONTOLOGIES

Basically, an OWL ontology consists of a set of class and property definitions that model ontology concepts (W3C, 2012). OWL treats properties as first class citizens, so that they are defined apart from classes. Properties specify their domain and their range. The domain of a property limits the individuals to which the property can be applied to. The range of a property limits the individuals that the property may have as its value. Therefore, in order to obtain a closed OWL ontology, we could think that an external reference is a property whose range references to a class that is not included in the OWL ontology. However, given that OWL and other ontology languages (Brickley & Guha, 2014; Connolly et al., 2001) allow the definition of properties apart from classes, the definition of inheritance relationships between properties, and the definition of other relationships between classes (e.g. equivalence, disjointness, and so on), the closure problem demands a special consideration in OWL ontologies. Next, a method to solve the closure problem on OWL ontologies is proposed that can also be applied to other ontology languages that consider properties as first class citizens. Such a method is based on the ontology closure method proposed in this paper, but it is adapted to the OWL features aforementioned.

First, an ontology will be represented as a set of triplets, inspired in the RDF model (Gandon & Schreiber, 2014), as the next figure illustrates. Figure 4.a shows an abstract ontology that consists of four classes (`C1`, `C2`, `C3`, `C4`). Figure 4.b shows the ontology represented by means of triplets. From now on, we will call `O-triplets` to such a representation. Attributes used on the triplets of `O-triplets` of Figure 4.b are `domain` to represent that a class is domain of a property, `range` to indicate that a class is range of a property, and `subclassOf` to show that a class is subclass of another. Other attributes that are not in the figure could be `equivalentClass` to state that a class is equivalent to another one, `disjointWith` to state that a class is disjoint with another one, `subpropertyOf` to state that a property is subproperty of another property, and so on. A similar approach can be found in (Noy et al., 2004), where *concept definitions* are introduced. Concept definitions are set of RDF triplets so that the class properties are expressed by means of RDF triplets.

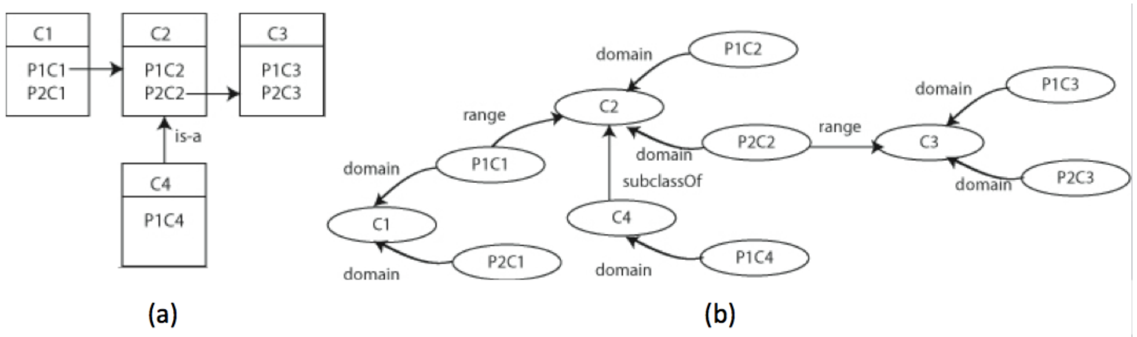


Figure 4: a) An abstract ontology; b) The same ontology represented by means of triplets (O-triplets)

In this model (`O-triplets`), an ontology is not closed in any of the following situations:

- A property has a reference to a property that is not included in the ontology (e.g. a property defined as inverse of a property that is not included in the ontology).
- A property has a reference to a class that is not included in the ontology (e.g. the range or the domain of a property is not included in the ontology).
- A class has a a reference to a class that is not included in the ontology (e.g. a class is equivalent to another class).

Applying closure to OWL ontologies basically deals with verifying whether `rdf:about` and `rdf:resource` attributes reference to components that are not included in the ontology. For enlargement closure, we only need to add every component referenced by the attributes `rdf:about` and `rdf:resource`. However, for reduction closure, it is not so easy, because the method proposed in the previous section for generic ontologies cannot be directly applied to OWL ontologies, as we will see next.

As we have described in Section 2.2, reduction closure allows obtaining closed ontologies, so that components that have external references are replaced with views that drop external references. This replacement may cause that other components that have (direct or indirect) references to replaced components must be updated. To reach OWL closure, as in the generic model described in Section 2, first we need to define the rules that determine which ontology components may be affected by the modification of a component. Next, a node list will be created, so that each node corresponds to a component to be modified. Each node will store the information needed to address the definition of the view component required to reach reduction closure. However, unlike the generic model, the rules for propagating changes, the building of the node list, and the information stored in each node, are slightly different to the ones proposed for the generic model. Finally, once the node list is built, components with external references, as well as components affected by change propagations, will be replaced with views so that a closed view ontology will be obtained.

3.1. OBTAINING THE COMPONENTS TO BE MODIFIED

In order to determine which are the components affected by the modification or elimination of an ontology component, the following set of rules have to be applied. This set uses a model for the ontology, named as `O-model`, that makes easier obtaining affected components. Such a model is built from the model `O-triplets`, described at the beginning of this section, following the issues shown below. This model is necessary because on the one hand, OWL considers properties a first class citizens; on the other hand, view properties may be defined from properties (Volz et al., 2003). Therefore, if a property is replaced with a view property in an ontology, the class which this property was applied to is also affected, and the remainder of properties of that class, as well as the remainder of ontology components directly or indirectly related with them.

- For each node of `O-triplets`, a node is built in `O-model`.
- If two nodes N_1 and N_2 of `O-triplets` are connected by means of a `range` relationship from N_1 to N_2, an arc from N_1 to N_2 must be built in `O-model`.
- If two nodes N_1 and N_2 of `O-triplets` are connected by means of a `domain` relationship from N_1 a N_2, a mutual arc between N_1 and N_2 must be built in `O-model`.
- If two nodes N_1 and N_2 of `O-triplets` are connected by means of an inheritance relationship (`subclassOf` or `subpropertyOf`) from N_1 to N_2, an arc from N_1 to N_2 must be built in `O-model`.
- If two nodes N_1 and N_2 of `O-triplets` are connected with another type or relationship (`disjointWith, inverseOf, ...`) from N_1 to N_2, a mutual arc between N_1 and N_2 must be built in `O-model`.

If these considerations are applied to the ontology expressed in `O-triplets` of the Figure 4.b, the graph of the next figure is obtained. In this graph, the rules proposed in Section 2.2.2 to obtain the components to be modified can be applied to.

In Figure 5, we show an example of how the components to be modified can be obtained. The example defines a view ontology from the ontology of Figure 4.a, but without including the class `C3` and its two properties. In this example, the components that have been selected to be included in the ontology are those depicted in Figure 6.a. In the figure we can see that the property `P2C2` has an external reference. If we apply to this model the rules for change propagation of Section 2.2.2, the class `C2` and the property `P1C2` will be affected. Besides, the subclass of `C2` (`C4`), as well as its property (`P1C4`), will be affected. Finally, given that the class `C1` has a property that has as range the class `C2`, then the class `C1`, and its properties, will be also affected by the changes. Therefore, all of these components must be replaced with views as illustrates Figure 6.b, which illustrates the ontology closed following the reduction approach.

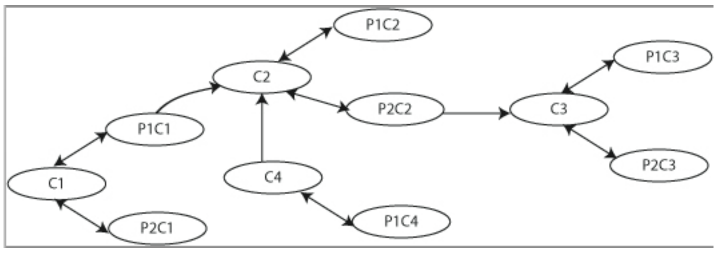


Figure 5: Adapted model of the ontology that makes easier to obtain which are the affected components (O-model)

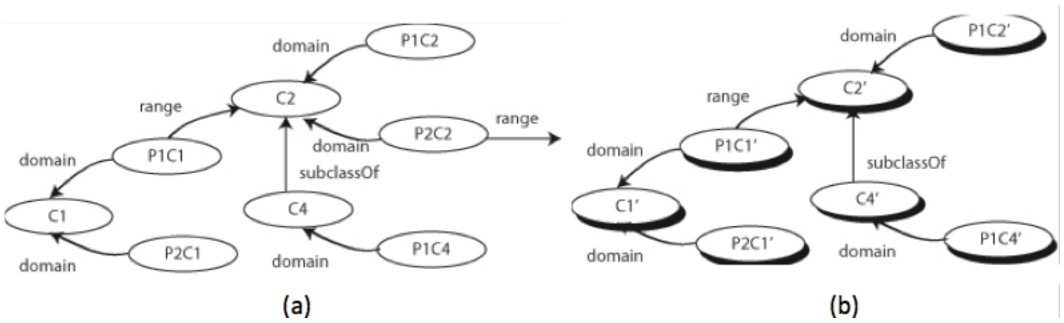


Figure 6: a) Components selected to make up a view ontology; b) View ontology closed following the reduction approach

As we see in Figure 6.b, the components with external references and the affected components have been replaced with views, which are represented using shaded ellipses. In the figure we also see that the names of the view components have also been modified. These new names represent that the view components are not the same that their base components. However, the only part of the component name that has to be modified is the base URI of the document where the definition of the view is done, but not the fragment part of the component URI. That is, base components and view components can have the same fragment part in their respective URIs but different base URI.

3.2. AN ILLUSTRATIVE EXAMPLE TO OBTAIN A CLOSED OWL ONTOLOGY

In order to illustrate how reduction closure can be achieved in an OWL ontology, let us suppose that we want to define a view ontology from the extended tourism ontology that we have already used in this work. This view was already defined using the generic model in Section 2.4. The view does not include the concepts `Contact`, `Acommodation`, and their details.

If reduction closure is the followed approach, first of all, the ontology must be represented in the triplets model proposed previously. Figure 7 illustrates the ontology expressed in `O-triplets`. In the figure, we can see that the ontology is not closed because properties `hasContact` and `hasAccommodation` has as range a class that has not been included in the view.

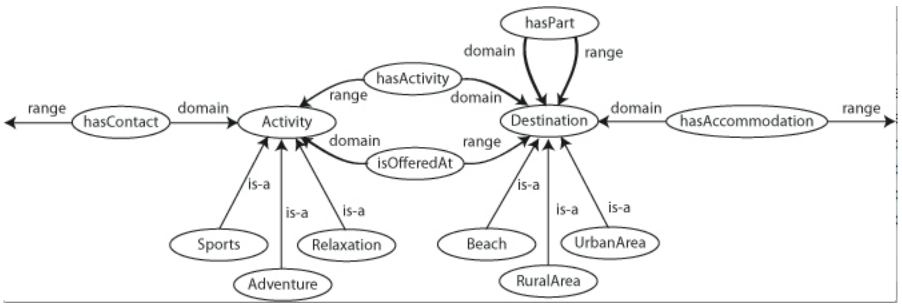


Figure 7: View ontology definition expressed in triplets

Second, the ontology expressed in `O-triplets` must be expressed in `O-model`, so that the following graph is obtained. Next, the rules for propagating changes are applied to this graph, and we obtain that given that `hasAccommodation` and `hasContact` have external references, `Destination` and `Activity` will be affected, as well as their respective subclasses and properties `hasActivity`, `isOfferedAt` and `hasPart`.

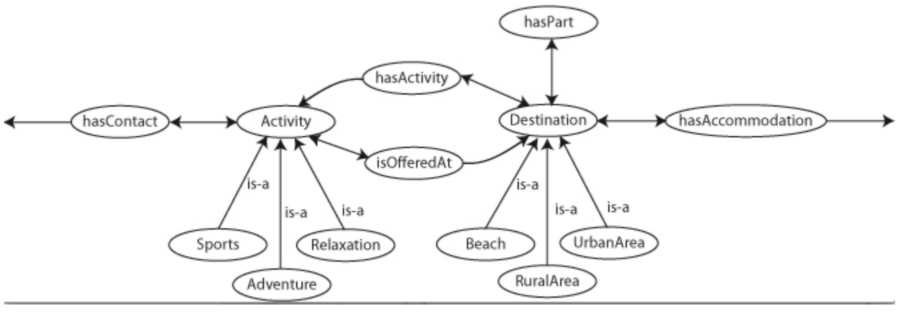


Figure 8: Ontology adapted model for applying the proposed closure algorithms

Following the reduction closure algorithm, the result of applying the rules for change propagation is a node list, which contains a node for each component with external references and a node for each component to be modified. In Section 2.2.2, a node structure for applying reduction closure to the generic ontology model was proposed. For OWL ontologies, the node structure differs slightly, because a new field (`ClassOrProperty`) has been added to the node to indicate whether it corresponds to a class or a property, so the node structure illustrated in Figure 9.a is obtained. This field will be used later by the view definition mechanism to define a view property or a view class. Figure 9.b illustrates the list initialized with the nodes that have external references (`hasContact` and `hasAccommodation`). Figure 9.c illustrates the resulting node list where the components referenced by each node and the list of updates is stored in the nodes. For example, the node corresponding to the class `Sports` shows that this class has an external reference to `Activity`. Given that the class `Activity` will be replaced with a view class `Activity1`, as is depicted in the node corresponding to the class `Activity`, the reference to the class `Activity` has to be updated to `Activity1`. Analogously, the node corresponding to the property `hasPart` shows that this property has a reference to `Destination` that must be updated to `Destination'`.

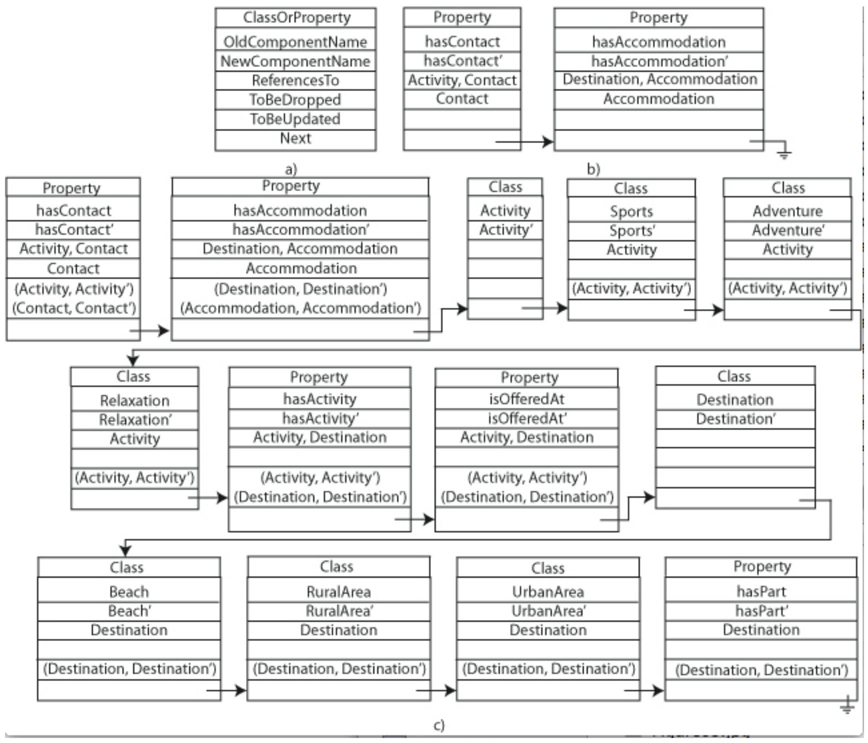


Figure 9: a) Node structure corresponding to components to be modified; b) Node list corresponding to components with external references; c) Final node list where propagations have been applied and references to new components have been updated to make easier the definition of view components

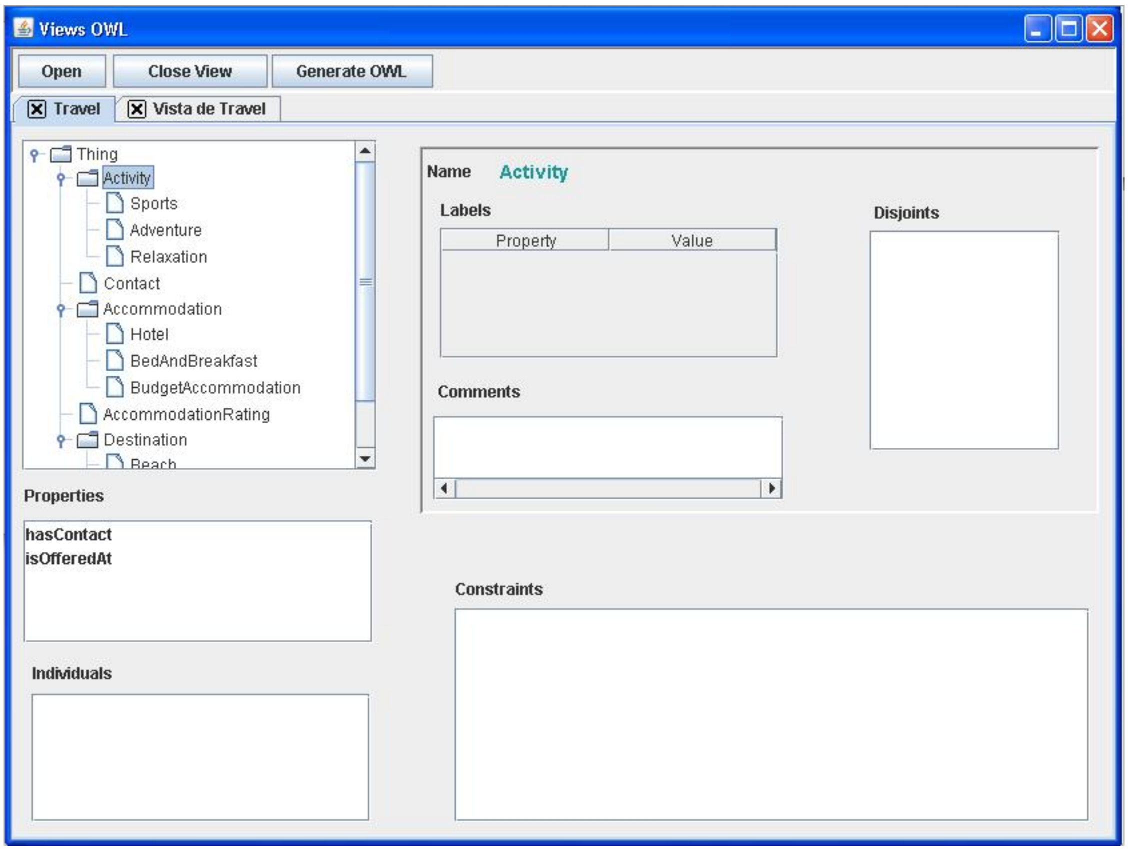
Once we have obtained the list of components to be modified, the ontology components corresponding to each node of the list must be replaced by view components so that the ontology is closed.

4. IMPLEMENTATION

We have developed a Java tool to implement the approaches proposed in this work for ontology closure. Our tool is based on a toolkit developed by IBM for ontology-driven development named IODT (Integrated Ontology Development Toolkit). This toolkit includes an Ontology Definition Metamodel (EODM), EODM workbench, and an OWL Ontology Repository, named Minerva, a high-performance OWL ontology storage, inference, and query system based on relational database management systems. EODM is derived from the OMG's Ontology Definition Metamodel (ODM) and implemented in Eclipse Modeling Framework (EMF).

The tool we have developed loads an OWL ontology, which will be stored in the OWL ontology repository. Next, the tool applies the closure approach selected by the user, and returns the results in an OWL ontology.

Next, we show how our tool can be used to apply reduction closure to the view ontology defined at the end of the previous section. The view was defined from the tourism ontology of Figure 1 we are following along this work, including activities, destinations and their subclasses, so that details related to contacts and accommodations are not included in the view. Figure 10 shows the components of the base ontology loaded in our tool. In the figure we can see some classes of the base ontology that are not going to be included in the view, like `Accommodation`. In the figure we can also see that in the base ontology, the properties of `Activity` are `isOfferedAt` and `hasContact`.



Name Activity

Properties
hasContact
isOfferedAt

Figure 10: Tourism ontology loaded in our closure tool

Figure 11 shows the effect of applying reduction closure to the defined ontology. This figure shows that components with external references, and affected components, have been replaced with views. For example, in the figure we can see that `Activity` has been replaced with `Activity1`, which does not have the property `hasContact`.

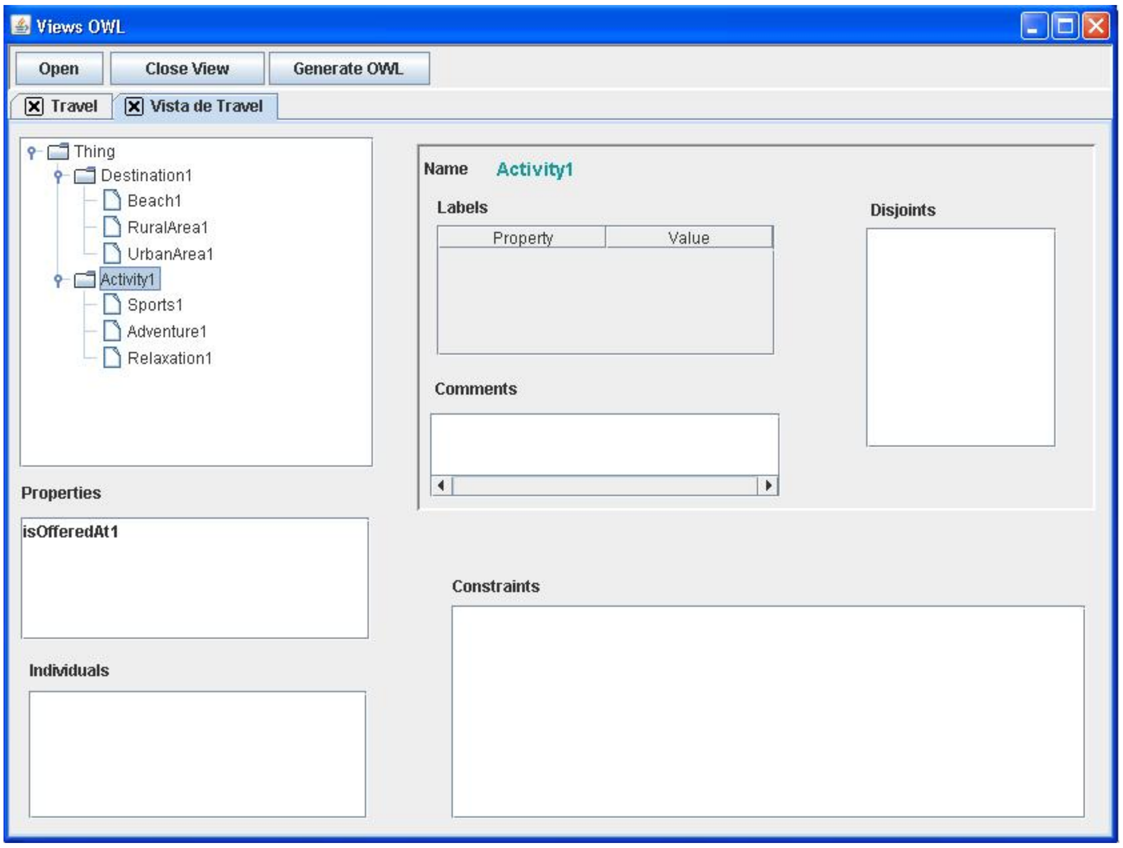


Figure 11: Resulting ontology after applying reduction closure in our tool

5. RELATED WORK

During the last years, several proposals have been developed to define view ontologies. Basically, most of them are mainly focused on the definition process of view classes and view properties in different languages, like OWL, RDF-S, and so on), or for generic ontology models. However, to the best of our knowledge, only a few proposals study the closure problem.

In (Uceda-Sosa et al., 2004), a language to define views on OWL components is proposed. The language is named as CLOVE and it extends OWL constraints so views may be defined. In CLOVE, a view specifies the objects it may be applied to, and the expressions that describe the view content. Views are considered as first-class citizens. CLOVE allows the definition of relationships between views (e.g. *is-a*). Views may be activated or in-activated by ontology owners, so that closure problems may emerge when users inactivate a view that is being referenced by another ontology component. In the paper, no mention is done about ontology closure.

In (Volz et al., 2003), a mechanism to define views for RDF-S ontologies is introduced. In the paper, a set-based operations to define object-preserving views using RQL query language (Karvounarakis et al., 2002) is proposed. View classes are placed in the class hierarchy when are defined as selection, difference, union or intersection views. The semantics of view definitions is used for such a purpose. If operations are combined, placement is not carried out automatically. In (Volz, 2003), this work is extended to define external ontologies (i.e., our view ontologies) in the sense of the well-known ANSI/SPARC three level architecture for databases, grouping view and base ontology components into new ontologies. However, the closure problem is not discussed.

In (Rajugan et al, 2005), an abstract view model for ontologies is proposed. The model is defined in UML using stereotypes to define conceptual views, and using OCL to specify view constraints. Ontology concepts are modelled as UML classes. Attributes and relationships are modelled as UML attributes and relationships, respectively. In the proposed model, views may be defined using set operators (union, intersection, cartesian product, join, and so on), and selection-projection operators. Nevertheless, ontology closure is not treated in that work.

In (Magkanaraki et al., 2004), RVL (RDF View Language) is introduced. RVL is based on RQL and it follows the RDF-S approach to consider properties as first-class citizens. RVL allows the creation of new classes and properties, as well as it allows the importation of existing components of another ontology. Views may be defined with object-preserving and object-generating semantics. Ontology closure is carried out by means of a restriction that states that the references of an ontology component (e.g. the range and the domain of a property) must always be defined. Thus, the creation of a view component is accompanied with the definition of its domain and range components.

In (Noy et al., 2004), views on ontologies are defined introducing the concept of *Traversal views* to reach ontology closure. Traversal views are views where users specify the central concept or concepts of interest, the relationships to traverse to find other concepts to include in the view, and the depth or maximum distance of the traversal. In this work, the *view boundary* concept is introduced to represent those classes that have not been included in the view ontology but are range of properties of classes included in the view ontology, that is, external references. In order to reach ontology closure, classes belonging to the view boundary are also included in the view ontology, so that enlargement closure is the used approach.

In (Seidenberg & Rector, 2006), an algorithm to obtain a subontology (i.e. view ontology) based on a set of classes selected by the user from an existing ontology is proposed. Needed classes are identified following the ontology link structure. The used approach is enlargement closure, so that components corresponding to external references are also included in the view. This work, like (Noy et al., 2004), proposes limiting the depth of the traversal, so that some referenced classes are in the boundary. That is, *boundary classes* are classes referenced by classes of the ontology, but that finally are not included in the view. However, if boundary classes are not included in the ontology, view components that reference them would have to be replaced with views because they do not represent the same concept any longer. Besides, this replacement would have to be propagated along the view, as we propose in this paper.

In (Volz, 2003), a view formalism for Semantic Web is proposed. The view formalism provides conceptual and logical semantics, as well as an extensive set of conceptual operators like union, intersection, join, projection, selection, and so on. Views are modelled using UML with stereotypes and constraints. The approach used to close the ontology is the enlargement approach, which is applied to the set of components selected to make up the view ontology, so that referenced components are also included in the view. In (Wouters et al., 2002), the closure process is illustrated by means of an example.

6. CONCLUSION

Views are an established technology for databases. However, views have not been studied so deeply in ontologies, although they are also needed (e.g. personalized access to metadata bases in community portals, authorization and integration of heterogeneous data sources). In (Magkanaraki et al., 2004; Rajugan et al, 2005; Uceda-Sosa et al., 2004; Volz et al., 2003; Noy et al., 2004; Volz, 2003), view mechanisms for ontologies are proposed. However, the *closure problem* is not studied in all the proposals. The closure is a property that must be satisfied in an ontology, so that every component referenced by a component of the ontology must also be included in the ontology. The interest of this problem grows when views may be defined on existing ontologies using any of the existing proposals. However, the set of components selected by the ontology definer to make up the view may be not closed. That is, some of the components used by some of the components of the view may be not included in the view ontology. Current works propose the use of *enlargement closure*, which enlarges view ontologies with the referenced components, so that the view ontology do not have external references.

In this work, a new approach to achieve ontology closure is proposed. The new approach, named *reduction closure* replaces the components that have external references with view components that hide those references. In addition, this replacement must be propagated and such a propagation may entail the definition of other view components. Reduction closure carries out this propagation automatically, simplifying the definition of view ontologies.

In this work, we have also shown how the closure concepts described in this paper may be applied to OWL ontologies. For such a purpose, we have introduced two intermediate models named `O-triplets` and `O-model`, allowing the application of the techniques proposed in this paper to achieve ontology closure in OWL ontologies. These models are needed because OWL, as well as other ontology languages, allow the definition of properties apart from classes, the definition of inheritance relationships between properties, and the definition of other relationships between classes (e.g. equivalence, disjointness, and so on).

Finally, we have shown a tool we have developed for applying the closure property to OWL ontologies.

REFERENCES

Brickley, D., & Guha, R.V. (2014). *RDF Schema 1.1.* Retrieved from *http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.*

Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P.F., & Stein, L.A. Annotated DAML+OIL ontology markup. (2001). Retrieved August 24th, 2014, from *http://www.w3.org/TR/2001/NOTE-daml+oil-walkthru-20011218/.*

Gandon, F., & Schreiber, G. *RDF 1.1 XML Syntax*. (2014). From *http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/*

Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., & Scholl, M. RQL: a declarative query language for RDF. (2002). In *Proceedings of the 11th international conference on World Wide Web*, 592–603. New York, NY: ACM.

Knublauch, H. *A tutorial ontology for a semantic web of tourism*. (2014). Retrieved August 24th, 2014, from *http://protege.cim3.net/file/pub/ontologies/travel/travel.owl.*

Lacroix, Z., & Delobel, C., & Brèche, P. (1998). Object views. *Networking and Information Systems*, 1(2-3), 231–250.

Magkanaraki, A., Tannen, V., Christophides, V., & Plexousakis, D. (2004). Viewing the Semantic Web through RVL lenses. *Journal of Web Semantics,* 1(4), 359–375

Mazón, J.N., & Trujillo, J. (2006). Enriching data warehouse dimension hierarchies by using semantic relations. In Bell, D.A., & Hong, J. *British National Conference on*

Databases. Proceedings of the 23rd International Conference (LNCS). (Vol. 4042, pp. 278–281). Berlin, Germany: Springer.

Mazón, J.N., Trujillo, J., Serrano, M., & Piattini, M. (2006). Improving the development of data warehouses by enriching dimension hierarchies with WordNet. In Collard, M. (Ed.), *Ontologies-Based Databases and Information Systems: Proceedings of the International Workshop (LNCS)* (Vol. 4623, pp. 85–101). Berlin, Germany: Springer.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K.J. (1990). Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, 3(4), 235–244.

Neumayr, B., Schütz, C., & Schrefl, M. (2013). Semantic Enrichment of OLAP Cubes: Multi-dimensional Ontologies and Their Representation in SQL and OWL. In Meersman, R., Panetto, H, Dillon, T., Eder, J., Bellahsene, Z., Ritter, N., …, Doo, D. (Eds.). *On the Move to Meaningful Internet Systems. Proceedings of the OTM 2013 International Conferences OTM 2013 (LNCS).* (Vol. 8185, pp. 624-641). Berlin, Germany: Springer.

Niemi, T., Toivonen, S., Niinimäki, M., & Nummenmaa, J. (2007). Ontologies with semantic web/grid in data integration for OLAP. *International Journal on Semantic Web Information Systems*, 3(4), 25–49.

Noy, N.F., & Musen, M.A. Specifying ontology views by traversal. (2004). In Mcllraith, S.A., Plexousakis, D., & van Harmelen, F (Eds.) *International Semantic Web Conference. Proceedings of the 3rd International Conference (LNCS)*. (Vol. 3298, pp. 713–725). Berlin, Germany: Springer.

Pardillo, J., & Mazón, J-N. (2011). Using ontologies for the design of data warehouses. International. *Journal of Database Management Systems,* 3(2), 73-87
Priebe, T., & Pernul, G. (2003) Ontology-based integration of OLAP and information retrieval. In *Proceedings of the 23rd International Workshop on Database and Expert Systems Applications (DEXA 2003),* 610–614. Prague, CA: IEEE Computer Society.

Rajugan, R., Chang, E. , Dillon, T., & Feng, L. (2005). Modeling ontology views: An abstract view model for semantic web. In Bramer, M., & Terziyan, V.Y. (Eds.) *Working Conference on Industrial Applications of Semantic Web. Proceedings of the First International Conference (IASW 2005)*, 188, 227–246. Berlin, Germany: Springer.

Romero, O. & Abelló, A. (2007) Automating multidimensional design from ontologies. (2007). In Song, I.Y., & Pedersen, T.B. (Eds.). *Data Warehousing and OLAP. Proceedings of the ACM 10th International Workshop on (DOLAP 2007),* 1-8. New York, NY: ACM.

Romero, O., & Abelló, A. (2007). Generating multidimensional schemas from the semantic web. In Eder, J., Tomassen, S. L. , Opdahl, A. L., & Sindre, G. (Eds.) *Conference on Advanced Information Systems Engineering: Proceedings of the 19th International Conference (CAiSE 2007), 247,* 11-15. Trondheim, Norway: CEUR-WS.org.

Romero, O., & Abelló, A. (2010). A framework for multidimensional design of data warehouses from ontologies. *Data Knowledge Engineering*, 69(11), 1138-1157

Rundensteiner. E.A. (1992). Multiview: A methodology for supporting multiple views in object-oriented databases. In Yuan, L-Y. (Ed.). *Very Large Databases. Proceedings of the 18th Internation Conference,* 187-198. Burlington, MA: Morgan Kaufmann.

Seidenberg, J., & Rector, A.L. (2006). Web ontology segmentation: analysis, classification and use. In *World Wide Web. Proceedings of the 15th international conference (WWW 2006),* 13–22. New York, NY: ACM.

Thomsen, E. (2002). *OLAP Solutions, Second Edition*. New York, NY: Wiley.

Torres, M., & Samos, J. (2001). Closed external schemas in object-oriented databases. In Mayr, H.C., Lazansky, J., Quirchmayr, G, & Vogel, P. (Eds.). *Database and Expert Systems Applications. Proceedings of the 12th International Conference (LNCS).* (Vol. 2113, pp. 826–835). Berlin, Germany: Springer.

Uceda-Sosa, R., Chen, C.X., & Claypool, K.T. (2004). CLOVE: A framework to design ontology views. In Atzeni, P., Chu, W., Lu, H., Zhou, S., & Ling, T.W (Eds.) *Conference on Conceptual Modeling. Proceedings of the 23rd International Conference (LNCS).* (Vol. 3288, pp. 844–849). Berlin, Germany: Springer.

Volz, R. External ontologies in the semantic web. (2003). In James, A.E., Lings, B., & Younas, M. *British National Conference on Databases. Proceedings of the 20th International Conference (LNCS).* (Vol. 2712, pp. 67-74). Berlin, Germany: Springer.

Volz, R., Oberle, D., Studer, R., & Staab, S. (2003). Views for light-weight web ontologies. *Symposium on Applied Computing. Proceedings of 2003 ACM SAC*, 1168–1173. New York, NY: ACM.
W3C OWL Working Group. (2012). *OWL 2 Web Ontology Language Document Overview (Second Edition)*. Retrieved from http://www.w3.org/TR/2012/REC-owl2-overview-20121211/

Wouters, C., Dillon, T.S., Wenny Rahayu, J., & Chang, E. (2002). A practical walkthrough of the ontology derivation rules. In Hameurlain, A., Cicchetti, R., & Traunmüller R. (Eds.). *Database and Expert Systems Applications. Proceedings of the 13th International Conference (LNCS).* (Vol. 2453, pp. 259-268). Berlin, Germany: Springer.

Wouters, C., Rajagopalapillai, R., Dillon, T.S., Rahayu, W. (2006). Ontology Extraction Using Views for Semantic Web. In Taniar D., & Rayahu W. (Eds.). *Web Semantics Ontology*, (1–40). Hershey, USA: Idea Group Inc.