A Method for Pruning Ontologies in the Development of Conceptual Schemas of Information Systems

Jordi Conesa and Antoni Olivé

Universitat Politècnica Catalunya
Departament de Llenguatges i Sistemes Informàtics
Jordi Girona 1-3 E08034 Barcelona (Catalonia)
{jconesa|olive}@lsi.upc.edu

Abstract. In the past, most conceptual schemas of information systems have been developed essentially from scratch. Currently, however, several research projects are considering an emerging approach that tries to reuse as much as possible the knowledge included in existing ontologies. Using this approach, conceptual schemas would be developed as refinements of (more general) ontologies. However, when the refined ontology is large, a new problem that arises using this approach is the need of pruning the concepts in that ontology that are superfluous in the final conceptual schema. This paper proposes a new method for pruning ontologies in this approach. We also show how to adapt the method to prune ontologies in other contexts. Our method is general and it can be adapted to most conceptual modeling languages. We give the complete details of its adaptation to the UML. On the other hand, the method is fully automatic. The method has been implemented. We illustrate the method by means of its application to a case study that refines the Cyc ontology.

1 Introduction

Most conceptual schemas of information systems have been developed essentially from scratch. The current situation is not very different: most industrial information systems projects are being developed using a methodology that assumes that the conceptual schema is created every time from scratch. However, it is well-known that substantial parts of conceptual schemas can be reused in different projects, and that such reuse may increase the conceptual schema quality and the development productivity [21].

Several research projects explore alternative approaches that try to reuse conceptual schemas as much as possible [5, 18, 29, 31]. The objective is similar to that of projects in the artificial intelligence field that try to reuse ontologies. There are several definitions of the term "ontology". We adopt here the one proposed in [12, 34], in which an ontology is defined as the explicit representation of a conceptualization. A conceptualization is the set of concepts (entities, attributes, processes) used to view a domain. An ontology is the specification of a conceptualization in some language. In this paper, we consider a conceptual schema as the ontology an information system needs to know.

S. Spaccapietra et al. (Eds.): Journal on Data Semantics V, LNCS 3870, pp. 64-90, 2006.

Ontologies can be classified in terms of their level of generality into[13]:

- Top-level ontologies, which describe domain-independent concepts such as space, time, etc.
- Domain and task ontologies which describe, respectively, the vocabulary related to a generic domain and a generic task.
- Application ontologies, which describe concepts depending on a particular domain and task.

We call top-level, domain and task ontologies general ontologies. One example of general ontology is Cyc [16].

General ontologies can play several roles in conceptual modeling [31]. One of them is the base role. We say that a general ontology plays a base role when it is the basis from which the conceptual schema is developed. In general, the development requires three main activities [10]: refinement, pruning and refactoring which are reviewed in section 3. The objective of the refinement activity is to extend the base ontology with the particular concepts needed in a conceptual schema, and that are not defined in that ontology.

In general, when the base ontology is large, the extended ontology cannot be accepted as the final conceptual schema because it includes many superfluous concepts. The objective of the pruning activity is then to prune the unnecessary concepts. In this paper, we propose a new method for pruning ontologies in the development of conceptual schemas. To the best of our knowledge, ours is the first method that is independent of the conceptual modeling language used and of the base ontology. The method can be used in other contexts as well, and we will show that it has several advantages over similar existing methods. Our method can be adapted to most languages, and we give the complete details of its adaptation to the UML [25]. We illustrate the method by means of its application to a case study that refines the Cyc ontology.

The structure of the paper is as follows. In the next section we present the case study used to exemplify our approach. Section 3 reviews the three main activities in the development of a conceptual schema from a base ontology, with the objective of defining the context of the pruning activity, the focus of this paper. Section 4 presents the pruning method we propose and proves it is correct. Section 5 compares our method with similar ones. Section 6 extends our method to make it independent of the selection strategy used to identify the concepts which are of direct interest for the information system. Finally, Section 7 gives the conclusions and points out future work.

2 Case Study

In the case study we create the conceptual schema of a recipe information system by refining the Cyc ontology. The information base must represent information about:

- *Recipes*: A recipe is a guide that explains how to create a given meal. They are published in documents written by one or more authors. Each recipe also indicates which ingredients are necessary to create the described meal for a given number of persons.

- *Ingredients*: A given quantity of an ingredient consists of one or more quantities of distinct nutrients.
- *Restaurants*: A restaurant is an organization whose main activity is to serve and prepare meals. Each restaurant offers a list of dishes available for a meal. The dishes are prepared by cookers. A restaurant can only offer the meals its cookers know prepare. Restaurants are located in cities. The name of a restaurant must be unique in the city where it is located.
- *Menus*: Restaurants offer menus, which are composed for a subset of the list of dishes. The menus must have at least one first dish, one second dish and one dessert. The price of a menu cannot exceed the addition of the individual prices per dish.

The information system must answer queries such as:

- Kilocalories of an ingredient.
- Amounts of lipid, carbohydrate, mineral, protein, vitamin, water and cholesterol an aliment has.
- For a given city, all the restaurants whose cookers have published a recipe.
- The recipe of a given meal with the lower number of calories.
- The restaurant of a given city that offers a given meal at the lowest price.
- All the vegetarian menus offered in a given city.
- For a given restaurant, the cheapest combination of first dish, second dish, and dessert.

More details will be given when they arise. The complete details of the case study are reported in [7].

3 The Context

In this section we briefly review the three activities required to develop a conceptual schema from a general ontology: refinement, pruning and refactoring. Normally, these activities will be performed sequentially (see Fig. 1), but an iterative development is also possible [10].

3.1 Refinement

Normally, a general ontology O_G will not include completely the conceptual schema CS required by a particular information system. The objective of the refinement activity is then to obtain an extended ontology O_X such that:

- O_X is an extension of O_G, and
- O_X includes the conceptual schema CS.

The refinement is performed by the designer. S/he analyzes the IS requirements, determines the knowledge the system needs to know to satisfy those requirements, checks whether such knowledge is already in O_G and, if not, makes the necessary extensions to O_G, thus obtaining O_X.

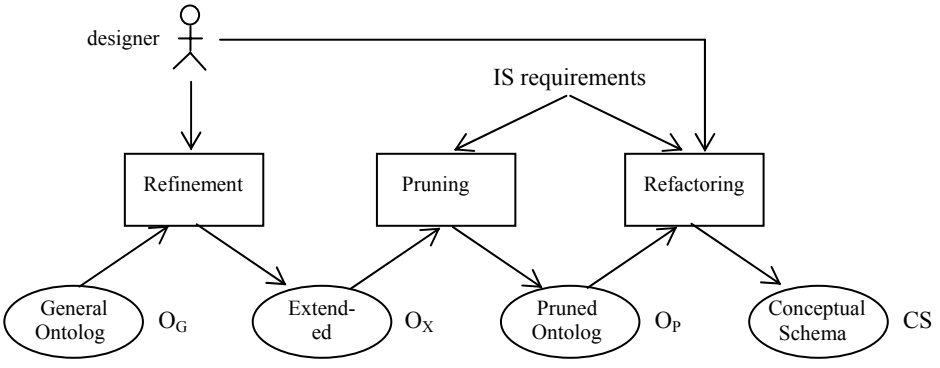

Fig. 1. The three activities in the development of conceptual schemas from general

In our case study, we adopted as general ontology O_G *OpenCyc* [26], the public version of the *Cyc* ontology. *OpenCyc* includes over 2900 entity types and over 1300 relationship types. Even if these numbers are large (and even larger in other ontologies such as *Cyc*) it is likely that additional entity or relationship types may be needed for the CS of a particular IS.

For example, our case study deals with recipes, its ingredients and the nutrients that compose those ingredients. The concept *Nutrient* exists in the base ontology, but their specializations into *Mineral*, *Lipid*... do not exist in OpenCyc. Then, we have to add a concept for each nutrient type: *Mineral*, *Lipid*, *Protein*, *Carbohydrates*, *Vitamin* and *Water-Ingestible* (see figure 2). Note that *Water-Ingestible* is also a *Drink*.

In our system, quantities of *EdibleStuff* must be expressed in some reference unit (such as gram). For this purpose we have defined attribute *referenceUnit* of type *UnitOfMeasure* (which is a datatype already defined in *OpenCyc*).

We need a concept that represents all kind of edible stuff element, because *EdibleStuff* represents also nutrients, and *Food* does not represent the condiments or preservatives that can be considered as ingredients. Then, we define an entity type called *NonNutrientEdibleStuff*. We define this type between *EdibleStuff* and its children: *CerealFood*, *FoodIngredientsOnly* and *FoodOrDrink*.

The nutritional composition of recipe ingredients is represented in the association between *NonNutrientEdibleStuff* and *Nutrient*. The association is reified in order to represent the quantity of nutrient included in the base quantity of *NonNutrientEdibleStuff*. For example *"100 gr. of rice have 7.3 gr. of proteins"*, where *rice* is an instance of *NonNutrientEdibleStuff* , with *baseQuantity* 100 gr., and the *nutrientQuantity* of proteins is 7.3 gr.

The complete refinement of *OpenCyc* for the case study is described in [7]. In summary, we have added twelve entity types (*Mineral, Lipid, Protein, CarboHydrates, Vitamin, Water-Ingestible, NonNutrientEdibleStuff, Recipe, RecipeDocument, FirstDish, SecondDish, Dessert, Menu, CateringCompany*), nine attributes (attributes *referenceUnit* of *EdibleStuff*, *baseQuantity* of *NonNutrientEdibleStuff*, *nutrientQuantity* of *NutritionalComponent* shown in Figure

2) and eight associations (one of them is *NutritionalComponent* in Figure 2). We have also added two association refinements and six general integrity constraints.

3.2 Pruning

Normally, an extended ontology O_X will contain many irrelevant concepts for a particular information system. The objective of the pruning activity is then to obtain a pruned ontology O_P such that:

- O_P is a subset of O_X, and
- O_P includes the conceptual schema *CS*, and
- The concepts in O_X but not in O_P would have an empty extension in the information system, or they are unnecessary for the information system.

In the case study, we find that the *OpenCyc* ontology contains thousands of concepts irrelevant for recipes. For example, the entity and relationship types dealing with Chemistry. Our information system is not interested in these concepts and, therefore, their extension in the information base would be empty. The objective of the pruning activity is to remove such concepts from O_X. In the next section we present a method for the automatic pruning of ontologies. The input of the method is either the formal specification of the IS requirements (domain events, queries) or the explicit definition of the concepts (entity and relationship types) of interest.

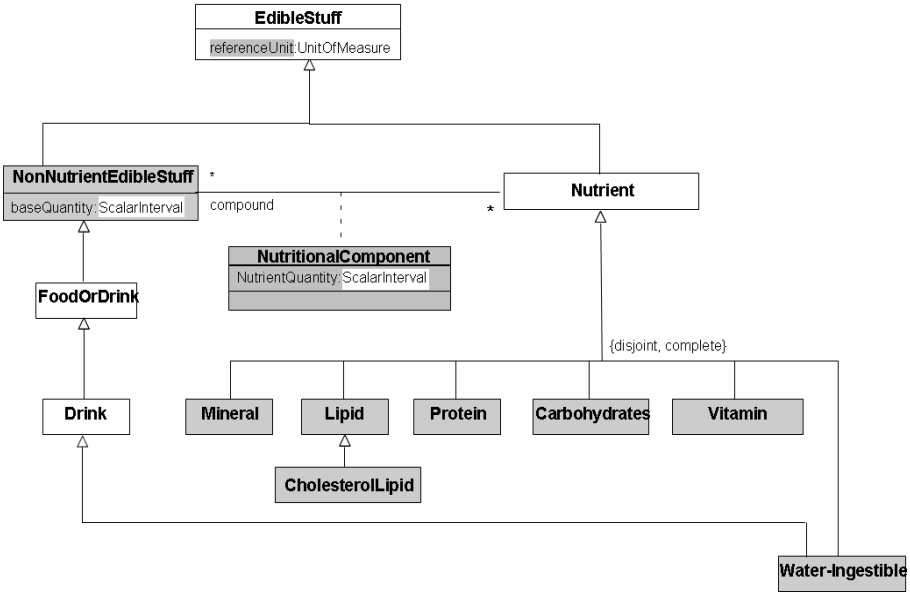

Fig. 2. Partial refinement of OpenCyc in the case study. The grayed boxes are refined concepts.

3.3 Refactoring

Normally, a pruned ontology O_P cannot be accepted as a final CS because it can be improved in several aspects. The objective of the refactoring activity is then to obtain a conceptual schema *CS* that is externally equivalent to O_P yet improves its structure. The purpose of ontology (or conceptual schema) refactoring is equivalent to that of software refactoring [11]. The refactoring is performed by the designer, but important parts of the activity can be assisted or automated, provided that the IS requirements are formalized. Refactoring consists in the application of a number of refactoring operations to parts of an ontology. Many of the software refactoring operations can be adapted to conceptual modeling, but this will not be explored in this paper.

4 Pruning the Extended Ontology

In this section, we define the problem of pruning the extended ontology and we propose a new method for its solution. The starting point of the pruning activity is an extended ontology O_X and the functional requirements of the IS. We explain also the adaptation of the problem and the method to the UML, currently one of the most widely used languages for conceptual modeling.

4.1 The Extended Ontology

We assume that, in the general case, an ontology O_X consists of sets of the following elements [33]:

- Concepts. There are two kinds of concepts:
  - Entity types.
  - Relationship types. We will denote by $R(p_1{:}E_1,\ldots,p_n{:}E_n)$ a relationship type R with participant entity types E_1, …, E_n playing roles p_1, …, p_n respectively.
- Generalization relationships between concepts. We denote by $IsA(C_1,C_2)$ the generalization relationship between concepts C_1 and C_2, where C_1 is the subtype and C_2 the super type. IsA^+ will be the transitive closure of *IsA*. We admit multiple specialization and multiple classification.
- Integrity constraints[1].

Adaptation to the UML. In the UML an ontology O_X consists of sets of the following elements (see Figure 2):

- Concepts:
  - Entity types.
  - Data types.
  - Attributes.
  - N-ary associations.

[1] The generalization relationships are (inclusion) constraints also, but we give them a special treatment due to its prominent role in taxonomies and in conceptual modeling.

  - Association classes, which reify associations. An association class and its reifying association are a single element.
- Generalization relationships between de above concepts. Attributes cannot be generalized.
- Constraints.

In the UML, some constraints are predefined (they have a particular language construct) and others may be user-defined. In our method we deal with constraints of the following kinds:

- Cardinality constraints of associations and attributes.
- Completeness and disjointness of sets of generalizations.
- Redefinition of association ends and attributes (redefinition constraints). Figure 3 shows an example of association redefinition: the association *ObjectFoundInLocation* is redefined in *City*.
- General constraints. We assume that general constraints are defined by constraint operations and specified in the OCL, as explained in [23]. The adaptation of our method to constraints defined as invariants is straightforward. An example is the constraint that the name of a restaurant must be unique into the city where it is located. Its formal specification is:

```
Context FoodServiceOrganization::uniqueName() : TruthValue
 body: FoodServiceOrganization.allInstances()->forAll(o1,o2|
           (o1 <> o2 and o1.name = o2.name) implies
                                            o1.City<>o2.City)
```

In the case study, O_X consists of:

- 2,715 Entity types and 255 Data types.
- 255 Attributes and 1,397 Associations.
- 6 general integrity constraints.

4.2 Concepts of Direct Interest

Usually, the extended ontology O_X will be (very) large, and only a (small) fraction of it will be needed for the CS of a particular IS. The objective of the pruning activity, as we will define it below, is to remove some non-needed elements from O_X.

The pruning activity needs to know which concepts from O_X are of direct interest in the IS. A concept is of direct interest in a given IS if its users and designers are interested in representing its population in the Information Base of the IS or inferring information from it. Our pruning method needs to know the concepts of direct interest, independently of how they have been obtained. We study in section 6 how to use several selection strategies to select the concepts of direct interest in an easy and reusable way.

When the functional requirements of an IS are formally specified, then the concepts of direct interest *CoI* may be automatically extracted from them [31]. The details of the extraction process depend on the method and language used for that specification. We explain here the process when the IS behavior is specified by system operations, as is done in many methods such as Larman's method [15], the B method [1] or Fusion [6]. A similar process can be used when the behavior is specified by statecharts, event operations or other equivalent methods.

In general, the formal specification of a system operation consists of:

- A signature (name, parameters, and result). The types of the parameters and the result are entity types defined in O_X.
- A set of preconditions. Each precondition is a boolean expression involving concepts defined in O_X.
- A set of postconditions. As above, each postcondition is a boolean expression involving concepts defined in O_X.

The concepts of direct interest *CoI* are then defined as:

- The types of the parameters and result of the system operations.
- The concepts appearing in the pre or postconditions.

In some cases the formal specification may not be available or may be incomplete. In these cases, the designers may wish to define the concepts *CoI* explicitly or to add new concepts to those determined from the functional specification.

If a relationship type is a concept of direct interest then we require that its participant entity types are in *CoI* also. Formally, we say that a set of concepts of direct interest *CoI* is *complete* if for each relationship type $R(p_1{:}E_1,\ldots,p_n{:}E_n) \in CoI$ the participant entity types $\{E_1, \ldots, E_n\} \subset CoI$.

In O_X there may be some concepts that generalize those in *CoI* and which are not part of *CoI*. We are interested in these generalized concepts because they may be involved in constraints that affect instances of the concepts *CoI*. To this end, we call set of *generalized* concepts of interest $G(CoI)$ the concepts of a complete set *CoI* and their generalizations. Formally:

$$G(CoI) = \{c \mid c \in CoI \vee \exists sub\ (\mathrm{IsA}^{+}(sub,c) \wedge sub \in CoI)\}$$

Adaptation to the UML. The adaptation is straightforward. We assume that the pre/postconditions are written in the OCL. For example, consider the system operation *howMuchCholesterol*, whose purpose is to return the quantity of cholesterol of a given meal. Its formal specification may be:

```
Context System::howMuchCholesterol (f:NonNutrientEdibleStuff):
                                                  NonNegativeNumber
  body: f.NutritionalComponent->
        select(nutrient.oclIsType(CholesterolLipid)).
                                          nutrientQuantity->sum()
```

The *CoI* inferred from this operation are: *NonNutrientEdibleStuff*, *NonNegativeNumber*, *NutritionalComponent*, *Nutrient*, *CholesterolLipid*, *NutrientQuantity* and *ScalarInterval*.

4.3 Constrained Concepts

We call constrained concepts of an integrity constraint *ic*, $CC(ic)$, the set of concepts appearing in the formal expression of *ic*. By abuse of notation we write $CC(O)$ to denote the set of concepts constrained by all the integrity constraints defined in ontology *O*. Formally,

$$CC(O) = \{c \mid c \text{ is a concept} \wedge c \in O \wedge \exists ic\ (ic \text{ is a constraint} \wedge ic \in O \wedge c \in CC(ic))\}$$

Adaptation to the UML. If *ic* is a cardinality constraint of an attribute or association, then *CC*(*ic*) will be the attribute or association, and the entity and data types involved in it.

If *ic* is a completeness constraint with a common supertype *super* and subtypes $sub_1, \ldots, sub_n$, then $CC(ic) = \{super, sub_1, \ldots, sub_n\}$.

A disjointness constraint with a common supertype *super* and subtypes $sub_1, \ldots, sub_n$, corresponds to $n(n-1)/2$ disjunction constraints each of which constraints two subtypes, sub_i and sub_j, and *super*. Strictly speaking, these constraints do not involve the supertype *super*, but in the UML they are attached to sets of generalizations having the same supertype.

If *ic* is a redefinition of an association or attribute then *CC*(*ic*) will be the redefined association or attribute, and the entity and data types involved in the association or attribute.

The constrained concepts of a general constraint will be the entity types, data types, attributes, associations and association classes appearing in the OCL expression that defines it. For example, if *uniqueName* is the general constraint defined in 4.1, and figure 3 represents the relevant fragment of the O_X for this integrity constraint, then CC(*uniqueName*) = {*FoodServiceOrganization, TruthValue, name, SomethingExisting, ObjectFoundInLocation, City*}. Note that the entity types *SomethingExisting* and *ObjectFoundInLocation* have been selected because they participate directly in the selected relationship types, which are *name* and *ObjectFoundInLocation* respectively.

4.4 The Pruning Problem

Given an extended ontology O_X and a complete set of concepts of direct interest *CoI*, as explained above, the pruning problem consists in obtaining a pruned ontology O_P such that:

(a) The elements in O_P are a subset of those in O_X. We do not want to add new elements to O_X in the pruning activity. Additions can be done in the refinement or in the refactoring activities.

$$\forall c\ (c \in O_P \rightarrow c \in O_X)$$

(b) O_P includes the concepts of direct interest *CoI*. These concepts must be included in O_P. The information system needs such concepts in order to perform its functions.

$$\forall c\ (c \in CoI \rightarrow c \in O_P)$$

(c) If C_1 and C_2 are two concepts in O_P and there is a direct or indirect generalization relationship between them in O_X, then such relationship must also exist in O_P. Otherwise, the child concept C_1 would lose the relationship types and constraints defined in C_2 or in its parents. Formally:

$$\forall c_1,c_2\ (c_1 \in O_P \wedge\ c_2 \in O_P \wedge \text{IsA}^+(c_1,c_2) \in O_X\ \rightarrow \text{IsA}^+(c_1,c_2) \in O_P)$$

(d) O_P includes all constraints defined in O_X whose constrained concepts are in $G(CoI)$. The rationale is that the constraints in O_X which constraint the Information Base of O_P must be part of it. The constraints in O_X that involve one or more concepts not in $G(CoI)$ cannot be enforced and, therefore, are not part of O_P.

$$\forall_{IC} ({}_{IC} \in O_X \wedge CC(_{IC}) \in G(CoI) \rightarrow {}_{IC} \in O_P)$$

(e) O_P is syntactically correct [17] , that is, it is a valid instance of the conceptual modeling language in which it is specified (metamodel).

(f) O_P is minimal, in the sense that if any of its elements is removed from it, the resulting ontology does not satisfy (b-e) above.

For each O_X and CoI there is at least an ontology O_P that satisfies the above conditions and, in the general case, there may be more than one.

To the best of our knowledge, there does not exist a method that obtains O_P automatically in a context similar to ours. In what follows we describe a method for the problem. In the next section we will compare it with existing similar methods.

4.5 The Pruning Algorithm

Our algorithm obtains O_P in three steps. The algorithm begins with an initial ontology O_0 which is exactly O_X (that is, $O_0 := O_X$) and obtains O_P. The steps are:

- Pruning irrelevant concepts and constraints. The result is the ontology O_1.
- Pruning unnecessary parents. The result is the ontology O_2.
- Pruning unnecessary generalization paths. The result is O_P.

Pruning irrelevant concepts and constraints. The concepts of direct interest for the IS are given in the set CoI, and $G(CoI)$ is the set of concepts in which the IS is directly or indirectly interested in. However, O_0 may include other concepts, which are irrelevant for the IS. Therefore, in the first step we prune from O_0 all concepts which are not in $G(CoI)$, that is, we prune the set of concepts:

$$\text{IrrelevantConcepts} = \{c \mid c \text{ is a concept } \wedge c \in O_0 \wedge c \notin G(CoI)\}$$

Pruning a concept C implies pruning of all generalization relationships $IsA(C_1,C)$ *and* $IsA(C,C_1)$ in which C participates. The super types and subtypes C_1 of C are not affected by the pruning of C.

Similarly, we prune the constraints in O_0 that are not relevant for the IS, because they constraint one or more concepts not in $G(CoI)$. That is, we prune the set of constraints:

$$\text{IrrelevantConstraints} =$$

$$\{ic \mid ic \text{ is a constraint} \wedge ic \in O_0 \wedge \exists c\, (c \in CC(ic) \wedge c \notin G(CoI)\}$$

The result of this step is the ontology O_1:

$$O_1 = O_0 - \text{IrrelevantConcepts} - \text{IrrelevantConstraints}$$

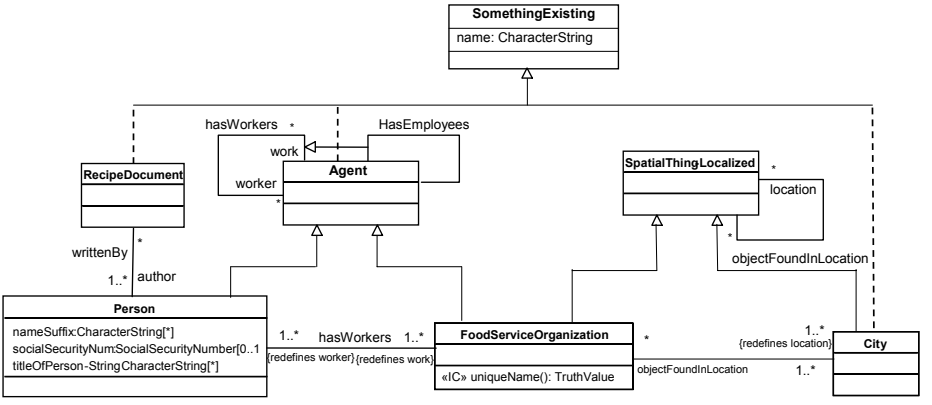

Fig. 3. Fragment of the extended ontology with relevant elements and constraints

In the example of Figure 3, we have that *HasWorkers* is a concept of interest and, therefore, $\{HasWorkers\} \subseteq G(CoI)$. However, *HasEmployees*, a subtype of *HasWorkers*, is not an element of $G(CoI)$ and therefore it is pruned in this step. Likewise, *Person* is a concept of interest but its subtypes (*Student*, *HumanChild*, *HumanAdult*, *FemalePerson*, *MalePerson*, etc. not shown in Figure 3) are not, and therefore they are also pruned in this step. The same happens to "lateral" concepts such as *Atom* or *Electron*.

In the case study, after the application of this step we have an ontology O_1 consisting of:

- 140 Entity types and 22 Data types.
- 15 Attributes and 30 Associations.
- 6 general integrity constraints.

Pruning unnecessary parents. After the previous step, the concepts of the resulting ontology (O_1) are exactly $G(CoI)$. However, not all of them are needed in the CS. The concepts strictly needed are given by:

$$\text{NeededConcepts} = CoI \cup CC(O_1)$$

The other concepts are potentially not needed. Formally:

$$\text{PotentiallyUnneededConcepts} = G(CoI) - \text{NeededConcepts}$$

We can prune the parents of *NeededConcepts* which are not children of some concept in *NeededConcepts*. Formally,

$$\text{UnnecessaryParents} = \{c \mid c \in \text{PotentiallyUnneededConcepts} \wedge \neg \exists c' (c' \in \text{NeededConcepts} \wedge \text{IsA}^+(c,c'))\}$$

As we have said before, the pruning of a concept implies the pruning of all generalization relationships in which that concept participates.

The result of this step is the ontology O_2:

$$O_2 = O_1 - \text{UnnecessaryParents}$$

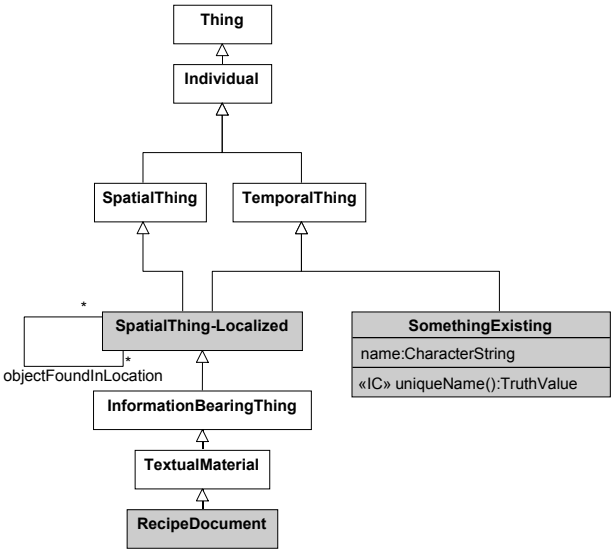

Fig. 4. Detecting and deleting the unnecessary parents. The grayed boxes are needed concepts

In Figure 4, examples of unnecessary parents are the entity types *SpatialThing*, *TemporalThing, Individual* and *Thing* which are the parents of *SpatialThing-Localized* and *SomethingExisting*. In the case study, *SpatialThing* neither is a needed concept of O_1, nor is a child of some needed concept, and therefore it is pruned in this step. The same happens for *Thing, Individual* and *TemporalThing*. Note that although the entity types *InformationBearingThing* and *TextualMaterial* are not unnecessary, they cannot be deleted, because of their common necessary parent *SpatialThing-Localized*.

In the case study, after the application of this step we have an ontology O_2 consisting of:

- 106 Entity types and 19 Data types.
- 15 Attributes and 11 Associations.
- 6 general integrity constraints.

Pruning unnecessary generalization paths. In some cases, the ontology O_2 may contain generalization paths between two concepts such that not all of them are necessary. The purpose of the third step is to prune these paths.

We say that there is a generalization path between C_1 and C_n if:

- C_1 and C_n are two concepts from O_2,
- $IsA^+(C_1,C_n)$ and
- The path includes two or more generalization relationships $IsA(C_1,C_2)$, …, $IsA(C_{n-1},C_n)$.

A generalization path $IsA(C_1,C_2)$, …, $IsA(C_{n-1},C_n)$ between C_1 and C_n is potentially redundant if none of the intermediate concepts C_2, …, C_{n-1}:

- Is member of the set $CoI \cup CC(O_2)$
- Is the super or the sub of other generalization relationships.

A potentially redundant generalization path between concepts C_1 and C_n is redundant if there are other generalization paths between the same pair of concepts. In this case, we prune the concepts C_2, …, C_{n-1} and all generalization relationships in which they participate. Note that, in the general case, this step is not determinist.

The output of this step is the pruned ontology, O_P.

Figure 5 shows four generalization paths between the concepts of direct interest *Restaurant* and *SomethingExisting*. None of these paths can be deleted, because at least one of their elements participate in more than one generalization relationship. Concretely, the entity types *FoodServiceOrganization*, *CommercialServiceOrganization*, *CommercialOrganization*, *LegalAgent, Organization* and *SocialBeing*. However, there exist three specialization paths between the entity types *Organization* and *FoodServiceOrganization*: P_1={*Organization*, *FoodAndBeverageOrganization*, *FoodServiceOrganization*}, P_2={*Organization*, *Service*, *FoodServiceOrganization*} and P_3={*Organization*, *CommercialOrganization, FoodServiceOrganization*}. The intermediate concepts of all the paths are not members of $CoI \cup CC(O_2)$. Furthermore, *FoodAndBeverageOrganization* is the only intermediate concept which does not participate in other generalization relationships, so the path P_1 is the only path that is potentially redundant. Therefore, it can be pruned from the ontology. After this, the algorithm will detect another duplicated specialization path between the concepts *Organization* and *CommercialServiceOrganization* composed by *{Organization, ServiceOrganization and CommercialServiceOrganization}*, and as a consequence the concept *ServiceOrganization* will be pruned.

In the case study, after the application of this step we have an ontology O_P consisting of:

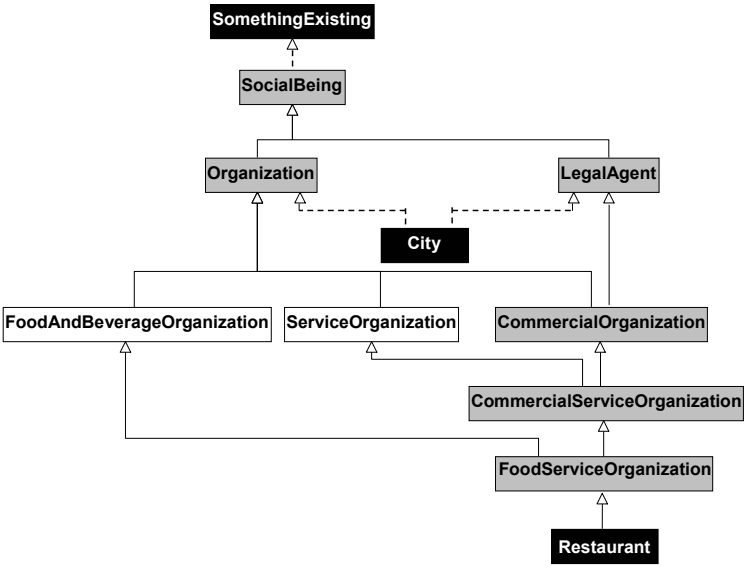

Fig. 5. Detecting and deleting the unnecessary duplicated paths between *Restaurant* and *Organization*. The white boxes are the concepts to prune and the black ones are the necessary concepts.

- 75 Entity types and 15 Data types.
- 15 Attributes and 11 Associations.
- 6 general integrity constraints.

4.6 Correctness of the Pruning Algorithm

In this section we argue that the above pruning algorithm is correct. We assume that the input to the algorithm is a syntactically correct extended ontology O_X and a set *CoI* of concepts of interest, with $CoI \subseteq O_X$. The pruning algorithm is correct if its output (the pruned ontology O_P) satisfies the conditions defined in section 4.4. In the following paragraphs we argue that O_P satisfies all of these conditions.

The elements in O_P are a subset of those in O_X. The algorithm only removes elements (concepts, constraints) from O_X. It never adds new elements. Therefore, in the general case, O_P will be a subset of O_X. In the rare case that all O_X concepts are of direct interest, then O_P and O_X would be the same.

O_P includes the concepts of direct interest *CoI*. None of the algorithm steps deletes any concept of direct interest:

- The pruning irrelevant concepts and constraints step removes the concepts not included in *G*(*CoI*). *G*(*CoI*) includes all the concepts of direct interest, therefore *CoI* concepts cannot be deleted in this step.
- The pruning of unnecessary parents step deletes a subset of the set of potentially unneeded concepts, which contains the concepts that are not of direct interest for the information system and do not appear in any relevant constraint. Obviously, this step cannot delete *CoI* concepts because of its exclusion of the potentially unneeded concepts set.
- The pruning unnecessary generalization paths step removes a subset of the potentially redundant elements set. This step cannot delete concepts of direct interest because that set does not include the *CoI* concepts.

Obviously, if no step can eliminate *CoI* concepts, then all *CoI* concepts will be included in the pruned ontology.

All O_P concepts with an *IsA* relationship in O_X, must also have an *IsA* relationship in O_P. None of the deletions done in the pruning steps results in a loss of specialization path between two needed concepts:

- The pruning irrelevant concepts and constraints step removes the *IsA* relationships relating irrelevant concepts. The irrelevant concepts neither are of direct interest, nor have children of direct interest, so we can affirm that *all the specializations of an irrelevant concept are also irrelevant*. As a consequence, whenever an irrelevant element is deleted, all its children are deleted as well. Then, it is obvious that none specialization path between survival elements may be deleted.
- The pruning of unnecessary parents step deletes the *IsA* relationships that relate unneeded elements, which are elements without necessary parents. Then, when the method deletes an unneeded element all its parents are deleted as well. As a consequence, it is obvious that their deletion do not break any specialization path.

– The pruning unnecessary generalization paths step removes the specialization paths between two concepts which satisfy a set of conditions, one of which is that the whole path is redundant. Then, eliminating a generalization path implies that there exists another generalization path between the same elements, so it is impossible to break a generalization path in this step.

Therefore, we can say our pruning activity does not delete necessary generalization paths between O_P concepts.

***O_P* includes all constraints defined in *O_X* whose constrained concepts are in *G*(*CoI*).** The pruning irrelevant concepts and constraints is the only step that deletes constraints. In particular, this phase only deletes the constraints whose concepts includes one or more irrelevant concepts, which are exactly the concepts not included into *G*(*CoI*). Then, for definition, we can conclude all the constrained concepts of the survival constraints are members of the set *G*(*CoI*).

***O_P* is syntactically correct.** An ontology is syntactically correct if all the constructions used to describe it are compliant with the grammar of its ontology language. For instance, an UML ontology is syntactically correct if it is a valid instance of the UML's metamodel and satisfies all its integrity constraints, including the well-formedness rules (WFR).

We assume O_X is syntactically correct. Then, in order to prove that O_P is also syntactically correct, we must prove that all possible deletions of the pruning method preserve the syntactic correctness of the ontology. In the following, we prove this for the deletion operations over the ontology elements:

– Concepts: Deleting a concept, implies deleting also all the generalization relationships where it participates. On the other hand, in our method, the deletion of a concept that participates in a given relationship or integrity constraint, also implies deleting its related concepts. Therefore, it is not possible to delete any concept that participates either in a relevant relationship type or a relevant integrity constraint.
– Generalization relationships: If a concept uses a relationship type defined in any of its parents, a deletion of a taxonomic relationship between the concept and its parent, may result in a syntactically incorrect ontology, because the child may lose the referred relationship type. Nevertheless, this particular case cannot occur in our algorithm, because, as we proved before, it does not allow breaking the generalization path between concepts.
– Integrity Constraints: They only restrict the possible instantiations of the ontology, so their deletion will result in a new ontology, less restricted, but not syntactically incorrect.

As a consequence, we can say that if the O_X is correct, the O_P will be correct as well.

***O_P* is minimal**. In order to prove that O_P is minimal, we are going to see which violations can be produced by the elimination of each kind of O_P element:

– Concepts: The concepts of the pruned ontology may be:
  – Concepts of Direct Interest: We cannot delete these concepts, because they must be included in the O_P (condition *b*). Their deletion may also produce the

violation of condition *c* (if the concept participates in one generalization and one specialization) or *e* (if any relationship type or constraint uses it).
  - Needed Concepts which are not *CoI*: These concepts are needed because they participate in one or more relevant constraints. Then, their deletion produces a syntactically incorrect ontology, because they are referred to in some relevant constraints.
  - Other concepts: These concepts are necessary to maintain a generalization path between, at least, two necessary concepts. Therefore, their deletion will break a non redundant specialization path, violating condition *c* of the method. Their deletion can also violate condition *e*.
- Integrity Constraints: The integrity constraints of the pruned ontology are those which can be evaluated using only elements of the *G*(*CoI*) set. Therefore, we cannot delete any of them without violating condition *d*.
- Generalization/specialization relationships. They are part of a non redundant generalization/specialization path between two (or more) necessary concepts. Obviously, we cannot delete any of them, without violating condition *c*.

Therefore, we have proved that the removal of any of the pruned ontology elements results in the violation of at least one condition of the pruning method.

5 Comparison with Previous Work

The need for pruning ontologies has been described in several research works in the fields of information systems and knowledge bases development. We may mention Swartout et al. [32], Knowledge Bus [27], Text-To-Onto [14, 19], Ontology Derivation Rules [39], MOVE [3, 4, 38], the ODS (Ontology-Domain-System) approach [36], DODDLE-II [30, 40], Mena et al. [20], the Dynamic Ontologies [28, 37] and OntoLearn [22]. In the following we explain the main differences among the pruning methods; we present a table that summarizes their main characteristics, and finally, give some comments and comparisons on the pruning methods which are more related to ours.

Even if the above works differ in the context in which the need for pruning arises, the ontology language, the particular ontology used as base, or the selection of the concepts of interest, we believe that (at least parts of) our pruning method can be adapted to be used successfully in all those works. The reason are: (1) we deal with any base ontology; (2) our method can be adapted to any ontology language (in [8] we show the adaptation of our method to the OWL (Web Ontology Language) [2]); (3) we take into account the specificity of entity types, relationship types, generalizations and constraints present in all complete conceptual modeling languages; and (4) although we obtain the concepts of interest from the functional specifications, our method can use any selection strategy to obtain the concepts of direct interest, as we will see in the next section.

Usually the pruning methods are intended to be applied in a particular context. The ontology context determines mainly its foremost properties: 1) the base ontology the method is able to prune, 2) how the method selects the concepts of direct interest, and 3) how many elements are pruned. The methods that use pruning techniques to support the information systems development, which are *Knowledge Bus*, *Ontology*

derivation rules, *MOVE*, *Ontology Domain System* and our method, allow pruning more expressive ontologies than the others. These methods also tend to do a more effective pruning, because their pruned ontologies are used directly for humans, and obviously, humans cannot deal easily with large ontologies (with more than a thousand concepts). Furthermore, those pruning methods, with the exceptions of *Knowledge Bus* and our method, have not been defined to prune very large ontologies. Examples are the *ODS* approach, which is totally manual, or the *Ontology derivation rules*, which works very well for small ontologies, but with too manual intervention to make it usable with large ontologies such as *OpenCyc*. In these methods, the user tends to participate very actively in the selection of the concepts of direct interest. The rationale is that in this context the user knows all the concepts that are relevant for the final information system and that must exist in the final ontology.

The goal of the other methods, which are *Swartout et al*, *Text-to-onto*, *Ontolearn* and *DODDLE-II*, is the creation of a domain ontology, whose information will be used to support users in a given task. These methods use linguistic ontologies as a basis, which are less expressive than those used by the above methods, but contain more concepts that the other ones. For example, SENSUS ontology, which is the ontology used by Swartout, et al., has more than 50,000 concepts, while *OpenCyc* does not have more than 5,000 concepts. These methods have more efficient selection processes, this is because they use the semantic relationships (synonyms, antonyms, ...) among concepts that the linguistic ontologies have. These methods are not interested in generating very small pruned ontologies, because their pruned ontologies are used for programs to infer information, and then, they should contain the concepts of direct interest and all their related concepts. For this reason, these pruning methods are equivalent to the first step of our pruning method, with the exception of DODDLE, whose pruning activity contains also a restructuring step.

Table 1 shows a few characteristics of the main current pruning methods. For each method we give: 1) the base ontology the method uses; 2) whether or not the method takes into account the integrity constraints (in one case we are unsure about this); 3) how automatic the method is; 4) the selection strategy used for selecting the concepts of direct interest; and finally, 5) the efficiency of the pruning activity, that is how many elements the pruned ontology has.

In the following we give some additional comments on the works which are the more closely related to ours, and that describe a comparable pruning method.

The purpose of the *Ontology Derivation Rules* is to generate domain or application ontologies using a set of rules over a base ontology. The base ontology can be any ontology written in the UML language. The designer is responsible of selecting the concepts of direct interest by-hand, which are called permanent elements in this method. The pruning method also restructures the ontology to minimize its volume. The restructuring is done by applying a set of rules, such as *when a non permanent class c1 contains a permanent attribute a1 and there is a permanent class c2 child of c1, then move the attribute a1 to c2*. After applying these rules, the method generates all the possible associations among the permanent classes of the ontology, following the associative property of the UML associations. Once all these hybrid relationships are created, the designer must identify the relevant ones. Finally, the method uses this information to delete the ontology irrelevant elements, and asks to the designer for a name for the survival hybrid associations. The results of this method are quite good, but its process is too manual to be usable with medium and large ontologies.

Table 1. Comparison of the main current pruning methods

	Base ontology	Integrity Constraints	Automation	Selection Strategy	Final Ontology
Knowledge Bus [27]	*OpenCyc*	✓	TOTAL	By-hand	TOO LARGE
Swartout et al. [32]	*SENSUS*, but applicable to any ontology	✗	SEMI-AUTOMATIC	By-hand	TOO LARGE
Ontology Derivation Rules [39]	Any UML ontology	?	SEMI-AUTOMATIC	By-hand	DESIRED
MOVE [3, 4, 38]	Any	Cardinality constraints	SEMI-AUTOMATIC	By-hand	CUSTOMIZABLE
Text-to-Onto [14, 19]	Any RDF ontology	Predefined in the language	SEMI-AUTOMATIC	Automatic: using text-mining algorithms	LARGE
OntoLearn [22]	WordNet	✗	SEMI-AUTOMATIC	Automatic: using text-mining algorithms	LARGE
Ontology Domain System [36]	Any UML ontology	✓	NONE	By-hand	DESIRED
DODDLE – II [30, 40]	WordNet	✗	SEMI-AUTOMATIC	By-hand	LARGE
Our method [9]	Any	✓	TOTAL	Any	CLOSE TO DESIRED

The reason is the high number of hybrid relationships that appears in its pruning process, and the hard work of the designers in identifying which is relevant and which is not. The same results or better can be obtained with the execution of our pruning and refactoring activities. For example, with reference to our case study, an ontology with over 4000 concepts and over 30 concepts of direct interest, the method will generate several hundreds of anonymous hybrid relationships (note that in an ontology with the magnitude of OpenCyc it can exits easily a chain of relationships relating almost all the entity types of the ontology). In addition, as far as we know, it does not exist the formal definition of the whole method, so it is unclear the efficiency of the pruning method for real cases. Furthermore, our method is more automatic and efficient than this one, because the restructuring activity is done after the pruning method.

MOVE uses the *ontology derivation rules* approach to generate a view of a given ontology that satisfies a set of requirements. The method can prune any kind of ontology written in the IOS language. This language allows to represent concepts, attributes, binary relationships and cardinality constraints over relationship types and attributes. The concepts of direct interest (called “selected”) are selected by-hand by the user. The user also has to select the concepts that cannot appear in the final ontology (called “unselected”). Then, the pruning is executed by taking into account four optimization schemas: 1) RCOS, which uses the ontology derivation rules to guaranty the final ontology satisfies the users requirements (the selection); 2) SCOS, which validates the semantic completeness of the ontology, that is, if a concept is defined using other concepts, we cannot delete the last without losing information of the former; 3) WFOS, which contains the rules that guarantees the syntactic correctness of the final ontology; and 4) TSOS, which guarantees the obtained ontology is the smallest that can be obtained. Up to now, as far as we know, only the two first phases have been defined, so their results are neither proved, nor guaranteed to be minimal. The pruning method may be customized by changing the given optimization schemas or adding new ones [38]. As in the previous case, this method can be compared to our pruning and refactoring activities together, and the same efficiency reasoning of the previous method can be applied also here.

The purpose of Swartout et al. is the development of specialized, domain specific ontologies from a large base ontology. The base ontology is SENSUS, a natural language based ontology containing well over 50,000 concepts. The elements of the ontology are only entity types and generalization relationships. The concepts of interest are assumed to be a set of entity types (called the "seed") selected explicitly by domain experts, and all entity types that generalize them. The pruning method corresponds roughly to our first step (pruning irrelevant concepts and constraints). Using our method, the domain experts could select the seed, as before, but also the generalized entity types of interest. The two other steps of our method could then be applied here, thus obtaining more specific domain ontologies.

The purpose of Knowledge Bus [27] is to generate information systems from application-focused subsets of a large ontology. The base ontology is Cyc, and the ontology language is CycL. The concepts of interest are assumed to be the set of entity types defined in a context (a subset of Cyc), also called the "seed" set, and all the entity and relationship types that can be "navigated" directly or indirectly from them. For example, with reference to Figure 3, if the seed set were only {*FoodServiceOrganization*} then all entity and relationship types shown in that figure would be considered concepts of interest. If we consider not only the fragment shown in that figure but the complete OpenCyc, then over 700 entity types and 1300 relationship types would be considered concepts of interest. The pruning method (called the sub-ontology extractor) corresponds here also to our first step (pruning irrelevant concepts and constraints). The result is that (as the authors recognize) many superfluous types are extracted from Cyc. Using our method, the domain experts can be more precise with respect to the concepts of interest. They could select the seed, as before, but also the generalized entity and relationship types of interest. The two other steps of our method could then be applied here as well, thus improving the specificity of the sub-ontology extraction process.

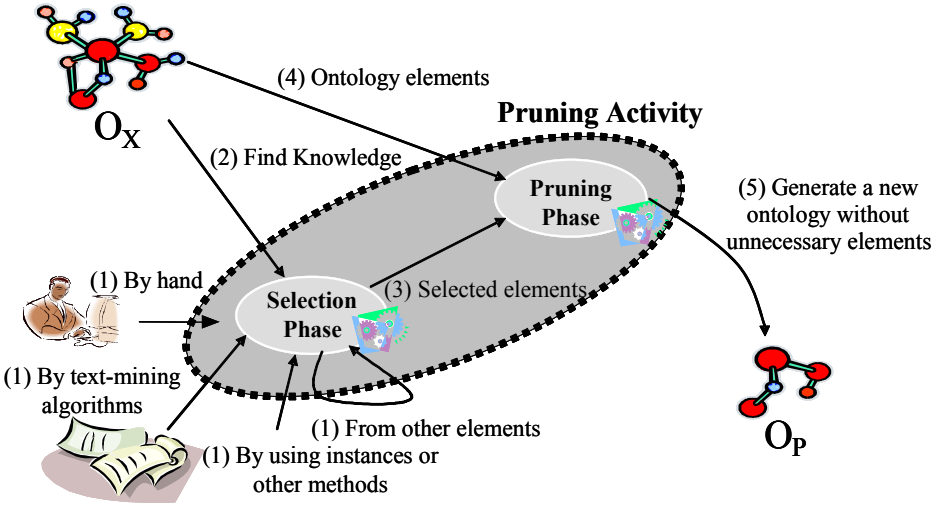

Fig. 6. Separating the pruning and selection phase in the pruning activity

6 Adapting Our Pruning Method to Different Selection Methods

Up to now, we have defined and solved the pruning problem in the context of developing the conceptual schema for an information system. In this section we show that our method can be used in other contexts as well.

Before pruning an ontology, it is necessary to select those elements that must be included in the final result. In our context, this selection is done using the information system requirements but in other contexts other selection strategies may be more suitable. For example, in the semantic web it can be necessary to select the elements using text mining algorithms [19], or manually [20]. In many methods, the selection of elements is an integral part of the pruning process. This implies that the selection strategy cannot be changed without re-implementing the pruning process. Here, we propose to separate the phases of selection and pruning (figure 6). This will allow us to do the pruning activity applicable for any strategy selection and able to reuse other selection methods.

In what follows we present a taxonomy (summarized in Figure 7) that describes the different ways of concept selection. Then we study how to use the taxonomy to reuse selection methods written by others. Finally we use the taxonomy to classify the main actual pruning methods.

6.1 Taxonomy of Relevant Concepts Selection Methods

According to its granularity, selection methods may be classified as individual or composite selection. An **individual** selection (also known as primitive selection) computes a selection based on a single selection criteria, and may be classified into

manual or automatic selection. In the manual selection, the designer must select by hand the elements of O_X that are necessary to the final ontology. The manual selection may be classified into:

- Unassisted selection: this is the most usual selection method. The designer chooses the necessary concepts without any system assistance. This method is used in [3, 9, 27, 32, 38, 40], where the designer selects manually the set of concepts relevant for the final ontology.
- Assisted selection: The system supports the user by proposing concepts to select. This kind of selection is usually combined with other selection methods (composite selection). We can see an example in the last step of the Swartout et al. approach [32], in which the system may propose the selection of ontology subtrees.

In the automatic selection, the concepts of direct interest are selected automatically by the system. This kind of selection must use some information to detect automatically new concepts of direct interest. This information can be taken from:

- Other selected concepts: The concepts of direct interest previously selected are used to select new concepts. An example of this kind of selection can be seen in [27], where the set of selected classes (CoI) is used to obtain all the relationships applicable to the classes of the CoI set (that is, the relationships whose participants are contained into CoI).
- Other ontology elements: Sometimes the non concept elements of the ontology (the ones that are not entity types or relationship types: individuals, classification relationships, …) are used to select new concepts. This is one of the most forgotten techniques of selection on pruning algorithms, but we think that it may be interesting in some cases to obtain the concepts of direct interest from the instances of the ontology, its integrity constraints, or its generalization relationships.
- External sources: The concepts of direct interest may also be obtained from information that lies in external sources. This is one of the most common techniques to select concepts of direct interest in pruning algorithms. Examples of this kind of selection are [22, 35], where the concepts of direct interest are obtained applying text-mining algorithms to several documents. There is another example in

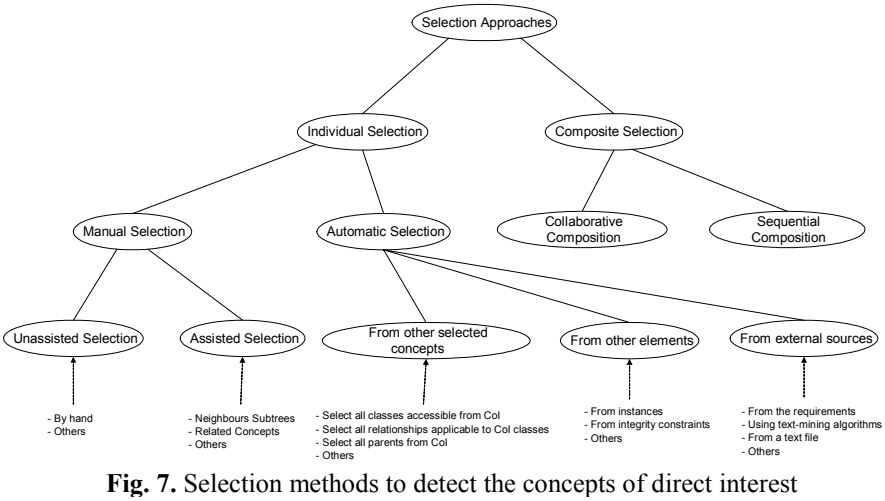

Fig. 7. Selection methods to detect the concepts of direct interest

the case study of this document, where the concepts of direct interest are detected automatically from the requirements of the IS, formalized by means of system operations [15] written in OCL [24].

Composite selection: A composite selection method includes more than one selection approach (that can be individual or composite). A composite selection may be:

- Collaborative composition: Several selection approaches are used collaboratively to detect the elements of direct interest. In this approach the outputs of the different selection approaches are evaluated to determine which concepts to select. Although this technique is not used nowadays in the pruning activity, we think it provides a very powerful way to detect the concepts of direct interest. On the other hand it seems that this selection technique needs a high participation of the ontology designer to define which elements to select, and this may be a drawback in the pruning of large ontologies.
- Sequential composition: A sequential composition is composed of a sequence of selection approaches, in which the output of each selection approach is the input of the next one. This technique is the most used at the moment. An example of this approach is Swartout et al. [32], where the selection process is a sequential composition of three individual selections: 1) a manual selection where the user selects without assistance a set of concepts of direct interest, 2) an automatic selection that selects all the parents of the elements selected in the previous process, and finally 3) a neighbour subtrees selection where the user can select subtrees whose neighbours have been selected in the previous steps.

6.2 Allowing General Purpose Selection

Current pruning approaches do not separate the selection and pruning phase. Therefore, the pruning methods are hooked to a selection strategy, which cannot be changed without re-implementing the pruning method. The problem grows when the pruning algorithm is specific to a selection strategy or a base ontology (its language or its structure). For example, a non generic pruning algorithm may contain a rule like "*delete a concept when none of its synonyms has been selected as relevant*". This rule is part of a selection strategy, in fact we may classify this rule in our taxonomy as a selection from other selected component. In addition, a strategy selection tends to be dependent to a given ontology. In the example the use of the synonym relationship, which is particular of linguistic ontologies, makes the pruning algorithm not applicable to all ontologies.

Separating the selection and the pruning phase makes the pruning algorithm more concise and independent of both selection strategies and the ontology used. In the previous example we may state the previous rule in the selection phase "*select the synonyms of the relevant elements*", and the pruning phase will contain a rule like "*delete the non relevant elements*". It is obvious that this way of defining a pruning algorithm is more generic than the previous one.

This separation reports also reusability benefits, because it allows to reuse individual selection approaches defined and implemented by others. To define a composite selection strategy, an ontology designer has to obtain the primitive methods (reusing them or developing them from scratch) needed in the composition,

and write a program that executes these primitive methods sequentially, giving the result of each method to the next one, and finally returning the results of the selection to the pruning phase.

Now that a taxonomy of selection is defined (see figure 7), it is possible to define a framework that supports the designer in the definition of selection strategies. A selection strategy, which combines several kinds of selection strategies, may be specified by means of a high level language based on the selection taxonomy.

We say our pruning method is generic, because the set *CoI*, which is necessary to our pruning activity, may be obtained as a result of applying any selection strategy that could be expressed as an instance of the presented taxonomy.

6.3 Expressing the Actual Pruning Methods as a Combination of Primitive Selection Methods

We think our taxonomy is complete with regards to the pruning methods defined until now in the literature. In order to validate this affirmation we show in this subsection how the selection phase of the main pruning methods can be expressed as an instance of our taxonomy.

Knowledge Bus

The selection begins with the selection by-hand of the relevant classes. Then, the system executes a fix point algorithm that selects all the classes that can be accessed from the relevant classes following relationships. Finally, all the associations whose participants have been selected in the previous steps are selected as well.

It is easy to see that the knowledge bus selection strategy may be represented by a Sequential Composition of: 1) an unassisted by hand method that selects the classes of direct interest (*CoI*). 2) An automatic selection that obtains the classes accessible from the *CoI* classes through relationships (*Select all the classes accessible from CoI*), and 3) another automatic selection that selects all the relationships whose participants were selected in the previous steps (*Select all relationships applicable from CoI*).

Swartout et al.

In this approach the relevant concepts for the target domain are manually selected by the user. Then, for each selected concept, the system automatically selects the elements contained in the path defined between the root of the ontology and the concept. After that, the designer may select some subtrees of the ontology such that almost all their neighbours (concepts with the same parents) have been selected, assuming that if all the neighbours of a concept have been selected, then the concept probably must be selected as well.

This selection process can be defined as a sequential composition of: 1) an unassisted by hand method that selects the concepts of direct interest, 2) an automatic selection that uses the previous one to obtain all the parents of the selected concepts (*Select all parents Of*), and 3) an assisted selection that assists the designer to select the needed ontology subtrees whose neighbours have been selected (*Neighbour Subtrees*).

Note that the first step is the same that the first step in Knowledge Bus, so both approaches may reuse the same implementation of the primitive selection method *by hand*.

Our Approach
The selection process of our method can be defined as a sequential composition of: 1) an automatic selection that selects all the concepts referred to in the requirements of the IS (*From the Requirements*), and 2) an unassisted by hand method that selects the rest of concepts necessary to the IS that were not selected in the previous step (this might be the same method used in Knowledge Bus and Swartout et al. approaches).

Due to space limitations we cannot define here all the pruning methods in terms of our taxonomy, but the application to the other pruning approaches is straightforward.

7 Conclusions

We have tried to contribute to the approach of developing conceptual schemas of information systems by reusing existing ontologies. We, as many others, believe that this approach offers a great potential for increasing both the conceptual schema quality and the development productivity.

We have focused on the problem of pruning ontologies. The problem arises when the reused ontology is large and it includes many concepts which are superfluous for the final conceptual schema. The objective of the pruning activity is to remove these superfluous concepts.

We have presented a new formal method for pruning an ontology. The input to our method is the ontology and the set of concepts of interest. When the functional requirements are formally specified, the concepts of interest can be automatically extracted from them. From this input, our method obtains automatically a pruned ontology, in which most of the superfluous concepts have been removed. We have shown that the method is correct.

We have formalized the method independently of the conceptual modeling language used. However, the method can be adapted to most languages, and we have shown the details of its adaptation to the UML. A prototype to prune UML ontologies has been implemented[2]. The adaptation of the pruning method to the OWL [2] is described in [8]. On the other hand, our method can be used with any ontology. The method has been illustrated by means of its application to a case study that refines the public version of the Cyc ontology. Our method improves on similar existing methods, due to its generality and greater pruning effectiveness.

The pruning method has been generalized in order to prune ontologies in other contexts. Changing the context of pruning application may result in a change of the way the concepts of direct interest are selected. Our pruning method is independent of the context, in the sense that it may be customized to be applied in any context and taking into account any way of selecting the concepts of direct interest.

[2] http://www.lsi.upc.es/~gmc/Downloads/jconesa/Program.zip

We plan to continue our work in (at least) two directions. First, we would like to implement our pruning method into a CASE tool. This will allow the designer to use the pruning method in a automatic and usable way. Finally, we plan to work on the activity that follows pruning: refactoring. The large amount of existing work on schema transformation can be "reused" for that purpose.

Acknowledgments

We would like to thank Jordi Cabot, Xavier de Palol, Dolors Costal, Cristina Gómez, Anna Queralt, Ruth Raventós, Maria Ribera Sancho and Ernest Teniente for their many useful comments to previous drafts of this paper.

This work has been partly supported by the Ministerio de Ciencia y Tecnologia and FEDER under project TIC2002-00744.

References

1. J. R. Abrial, *The B-Book*, 1996.
2. S. Bechhofer, F. v. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, *OWL Web Ontology Language Reference* , http://www.w3.org/TR/owl-ref/: W3C, December, 2003.
3. M. Bhatt, A. Flahive, C. Wouters, W. Rahayu, and D. Taniar, "A Distributed Approach to Sub-Ontology Extraction," in *Proceedings of the 18th International Conference on Advanced Information Networking and Application*. Fukuoka, Japan, 2004.
4. M. Bhatt, C. Wouters, A. Flahive, W. Rahayu, and D. Taniar, "Semantic Completeness in Sub-ontology Extraction Using Distributed Methods," in *Proceedings of ICCSA 2004*, 2004, pp. 508-517.
5. S. Castano, V. D. Antonellis, and B. Zonta, "Classifying and Reusing Conceptual Schemas," in *ER'92*, vol. 645, *Lecture Notes in Computer Science*, G. Pernul and A. M. Tjoa, Eds., 1992, pp. 121-138.
6. D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Guilchrist, F. Hayes, and P. Jeremaes, *Object-Oriented Development. The Fusion Method*: Prentice Hall, 1994.
7. J. Conesa, "Ontology Driven Information Systems Development: Pruning and refactoring of ontologies. PhD Thesis (In preparation)," in *LSI - Llenguatges i Sistemes Informàtics*. Barcelona: UPC, 2005.
8. J. Conesa and A. Olivé, "A General Method for Pruning OWL Ontologies," in *ODBASE'04*, vol. 3291, *Lecture Notes in Computer Science*. Larnaca, Cyprus, 2004, pp. 981-998.
9. J. Conesa and A. Olivé, "Pruning Ontologies in the Development of Conceptual Schemas of Information Systems," in *ER2004*, *Lecture Notes in Computer Science*. Shangai, China, 2004.
10. J. Conesa, X. d. Palol, and A. Olivé, "Building Conceptual Schemas by Refining General Ontologies," in *DEXA'03*, vol. 2736, *Lecture Notes in Computer Science*, 2003, pp. 693 - 702.
11. M. Fowler, *Refactoring: Improving the Design of Existing Code*: Addison-Wesley, 1999.
12. T. R. Gruber, "Toward Principles for the Design of Ontolgies for Knowledge Sharing," in *International Journal of Human and Computer Studies*, vol. 43 (5/6), 1995, pp. 907 - 928.

13. N. Guarino, "Formal Ontology and Information Systems," in *Proc. FOIS'98*: IOS Press, 1998, pp. 3-15.
14. J.-U. Kietz, A. Maedche, and R. Volz, "A Method for Semi-Automatic Ontology Acquisition from a Corporate Intranet," in *Proceedings of EKAW-2000 Workshop*, *Springer Lecture Notes in Artificial Intelligence (LNAI)*, 2000.
15. C. Larman, *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design*: Prentice Hall, 1998.
16. D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd, "CYC: Toward Programs With Common Sense," *Communications of the ACM*, vol. 33, pp. 30-49, 1990.
17. O. I. Lindland, G. Sindre, and A. Solvberg, "Understanding Quality in Conceptual Modeling," *IEEE Software*, vol. 11, pp. 42-49, 1994.
18. M. Lloyd-Williams, "Exploiting Domain Knowledge During the Automated Design of Object-Oriented Databases," in *Proc. ER '97*, vol. 1331, *Lecture Notes in Computer Science*, D. W. Embley and R. C. Goldstein, Eds.: Springer, 1997, pp. 16-29.
19. A. Maedche and S. Staab, "Ontology Learning for the Semantic Web," *IEEE Inteligent Systems*, vol. 16, pp. 72 - 79, 2001.
20. E. Mena, J. A. Royo, A. Illarramendi, and A. Goñi, "An Agent-based Approach for Helping Users of Hand-Held Devices to Browse Software Catalogs," in *In Cooperative Information Agents VI, 6th International Workshop CIA 2002*. Madrid, Spain, 2002.
21. H. Mili, F. Mili, and A. Mili, "Reusing Software: Issues and Research Directions," *IEEE TSE*, vol. 21, pp. 528-562, 1995.
22. R. Navigli, "Automatically Extending, Pruning and Trimming General Purpose Ontologies," in *International Conference on Systems, Man and Cybernetics*. Tunisy, 2002.
23. A. Olivé, "Integrity Constraints Definition in Object-Oriented Conceptual Modeling Languages," in *ER'03*, vol. 2813, *Lecture Notes In Computer Science*, 2003, pp. 349 - 362.
24. OMG, "OMG Revised Submission, UML 2.0 OCL,"
25. OMG, *UML 2.0 Superstructure Specification, 2.0 edition*: OMG, August, 2003.
26. OpenCyc, "OpenCyc, the public version of Cyc,"http://www.opencyc.com/
27. B. J. Peterson, W. A. Andersen, and J. Engel, "Knowledge Bus: Generating Application-focused Databases from Large Ontologies," in *KRDB'98*, *CEUR Workshop Proceedings*, 1998, pp. 2.1-2.10.
28. G. Sacco, "Dynamic Taxonomies: A Model for Large Information Bases," *IEEE Transactions on Data and Knowledge Engineering*, vol. 12, pp. 468-479, 2000.
29. V. C. Storey, R. H. L. Chiang, D. Dey, R. C. Goldstein, and S. Sundaresan, "Database Design with Common Sense Business Reasoning and Learning," *ACM TODS*, vol. 22, pp. 471-512, 1997.
30. N. Sugiura, M. Kurematsu, N. Fukuta, N. Izumi, and T. Yamaguchi, "A Domain Ontology Engineeering Tool with General Ontologies and Text Corpus," in *Proceedings of the 2nd Workshop on Evaluation of Ontology based Tools*, 2003, pp. 71-82.
31. V. Sugumaran and V. C. Storey, "Ontologies for conceptual modeling: their creation, use, and management," *Data & Knowledge Engineering*, vol. 42, pp. 251-271, 2002.
32. W. R. Swartout, R. Tatil, K. Knight, and T. Russ, "Toward Distributed use of Large-Scale Ontologies," in *Proc. 10th. Knowledge Acquisition for Knowledge-Based Systems Workshop, Canada*, 1996.
33. M. Uschold, "Knowledge level modelling: concepts and terminology," *The Knowledge Engineering Review*, vol. 13, pp. 5-29, 1998.
34. M. Uschold and M. Gruninger, "Ontologies: Principles, Methods and Applications," *The Knowledge Engineering Review*, vol. 11, pp. 93-136, June, 1996.
35. R. Volz, R. Studer, A. Maedche, and B. Lauser, "Pruning-based Identification of Domain Ontologies," *Journal of Universal Computer Science*, vol. 9, pp. 520-529, 2003.

36. X. Wang, C. W.Chan, and H. J.Hamilton, "Design of Knowledge-Based Systems with the Ontology-Domain-System Approach," in *Software Engineering and Knowledge Engineering*, vol. 859. Ischia, Italy: ACM Press, 2002, pp. 233 - 236.
37. D. Wollersheim and W. Rahayu, "Methodology For Creating a Sample Subset of Dynamic Taxonomy to Use in Navigating Medical Text Databases," in *Proceedings of the 2002 International Symposium on Database Engineering & Applications*, 2002, pp. 276-289.
38. C. Wouters, T. Dillon, W. Rahayu, E. Chang, and R. Meersman, "Ontologies on the MOVE," in *Proceedings of the 9th International Conference on Database Systems for Advanced Applications*, vol. 2973, *Lecture Notes in Computer Science*. Jeju Island, Korea, 2004, pp. 812 - 823.
39. C. Wouters, T. S. Dillon, J. W. Rahayu, and E. Chang, "A Practical Walkthrough of the Ontology Derivation Rules," in *DEXA'02*, vol. 2453, *Lecture Notes in Computer Science*, 2002, pp. 259-268.
40. T. Yamaguchi, "Constructing Domain Ontologies Based on Concept Drift Analysis," in *IJCAI-99. Workshop on Ontologies and Problem-Solving Methods*, 1999.