

Ontology-Driven Information Systems: Pruning and Refactoring of Ontologies *

Jordi Conesa Caralt

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

Abstract. This paper presents a method to develop conceptual schemas 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 pruned ontology cannot be accepted as a final conceptual schema because it can be greatly improved. Therefore, we use refactoring operations to solve this problem in order to obtain the final conceptual schema. The present document takes special attention to the pruning and refactoring activities, which are the core of my thesis.

1 Introduction

In general, most of the conceptual schemas (CS) of information systems (IS) are created from scratch, wasting a lot of time and resources. This thesis aims to define a new approach to develop CS of IS reusing the knowledge that exists in large ontologies.

An ontology is the explicit specification of a conceptualization in some language [1]. In our work, we consider a conceptual schema as the ontology that an information system needs to know.

The general trend towards software reusability has also had its impact on conceptual modelling, and there have been many efforts trying to reuse existing ontologies in the development of new conceptual schemas [2]. In particular, an ontology plays the base role if it is the basis from which the CS is derived, so the CS can be seen as a specialization of the ontology.

The structure of the paper is as follows. Next section defines the basic structure of our proposal. Section 3 presents an overview of our pruning method and its remaining work. Section 4 explains the general structure of our refactoring activity and points out its further work. Finally, section 5 states the relevance of our work, and its application to other areas.

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

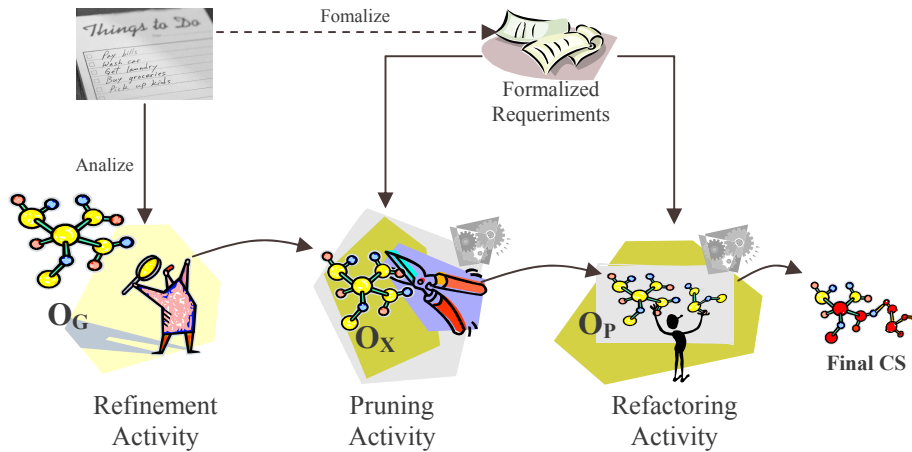


Figure 1: The structure of our proposal

2 The basic structure of our approach

Our proposal uses the base role to create the CS of an IS from a general ontology (O_G). Our proposal uses three activities to obtain the final CS (see figure 1):

- Refinement: O_G may not contain some of the elements (entity types, relationship types, attributes and integrity constraints) required for the IS. In this case, a phase is needed that allows the designer to execute a set of additive operations to the ontology to create those elements. Usually O_G will be quite large, so several techniques should be implemented to support its refinement in a usable and easy way. The result of this phase is what we call the extended ontology (O_X).
- Pruning: At this point, O_X has all the knowledge needed by the IS, but it cannot be used as CS because of its huge magnitude (it may have more than 10^3 entity types and 10^3 relationship types). The majority of the O_X elements are irrelevant to the IS, then a pruning phase is needed to delete these elements. After this phase, a pruned ontology is obtained (O_P).
- Refactoring: At this point we can say that all the concepts of O_P are relevant for the IS. However, some of them are redundant and must be deleted, and some other may be restructured (by refactoring operations) to create a CS more compact and with a higher quality. The result obtained is the final CS.

Due to the magnitude of the whole approach, my thesis is focused only in the pruning and refactoring activities. Our aim in this thesis is to define completely the two activities, automating them as much as possible.

A more exhaustive view of our approach can be obtained from [3]. Moreover, some examples of its application can be obtained from [4, 5].

3 Pruning Conceptual Schemas

As we have already seen, after refining O_G we obtain the extended ontology O_X . Usually, this ontology is not directly usable, because it is quite large and contains many irrelevant concepts for a particular IS. Thus, the pruning activity must identify and delete the O_X irrelevant concepts. We will use figure 2 to exemplify this activity.

3.1 Actual solutions to pruning ontologies

The need for pruning an ontology has been described in several research works in the fields of information systems and knowledge bases development. We may mention Swartout et al. [6], Knowledge Bus [7], Text-To-Onto [8], Wouters et al. [9], Yamaguchi [10] and OntoLearn [11].

Most of the above methods (Swartout et al., Text-To-Onto, OntoLearn and Yamaguchi) are used to prune linguistic ontologies. These ontologies do not contain some of the elements of the UML ontologies, such as integrity constraints (IC). We think the IC are fundamental in conceptual schemas, so we want a pruning method capable of dealing with them. Thus, we cannot use this pruning approaches.

The Wouters et al. approach [9] defines a method to prune UML ontologies. However, it has not been formally defined and merges both its pruning and refactoring phases, doing the method less general and modifiable. Thus it is not applicable to our proposal.

The Knowledge Bus [7] fulfills some of our requirements because allows pruning more expressive ontologies and takes into account the integrity constraints. Nevertheless, its pruned ontologies are too large to be used in our approach because of its pruning criteria.

3.2 The problem

The pruning activity needs to know which elements from O_X are of direct interest in the final ontology. A concept (for us a concept is either an entity type, a relationship type or an attribute) is of direct interest in a given ontology if the designers are either interested in representing its population, or inferring new information from it. We denote by CoI the set of concepts of direct interest in O_X .

The objective of our pruning activity is to obtain a pruned ontology O_P such that:

- a) O_P is a subset of O_X , and
- b) O_P includes the concepts of direct interest, and
- 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 , and
- d) If c is a concept, i is an instance in O_P and there is an *instanceOf* relationship between them in O_X , then such relationship must also exist in O_P , and
- e) O_P includes all constraints defined in O_X whose constrained concepts are either of direct interest, or parent of a concept of direct interest, and
- f) O_P is consistent, that is, it is a valid instance of the ontology modeling language in which it is specified, and

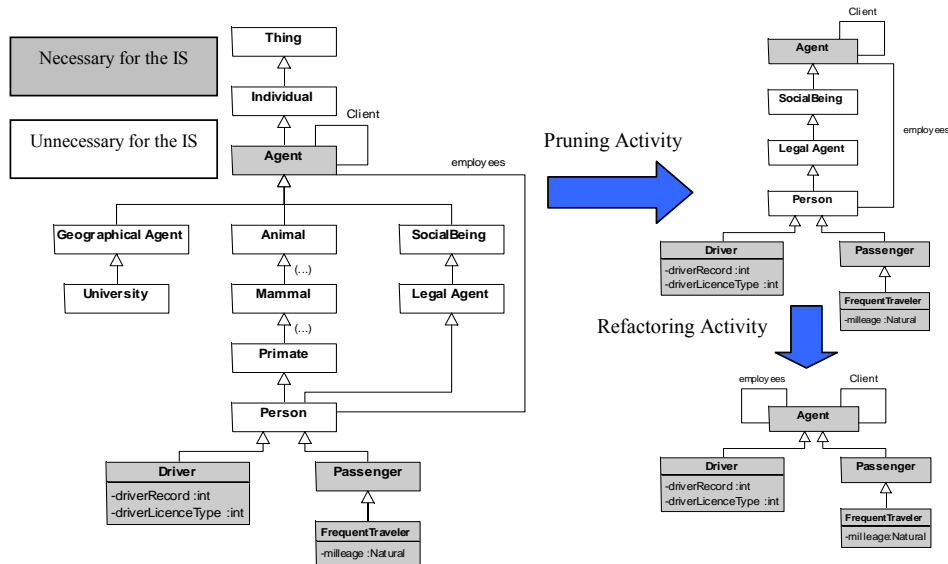


Figure 2: Example of pruning and refactoring activities

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

3.3 Our proposal

Our pruning method uses the functional requirements of the IS to select the concepts of direct interest. We assume that functional requirements are formalized as system operations [12], which consist of a signature, a precondition and a postcondition formalized in OCL.

The selection algorithm selects the concepts that are referred to within the functional requirements. When a relationship type or an attribute are selected, all their participants are selected as well. Then, the pruning algorithm obtains O_P in the following four steps.

- *Pruning irrelevant concepts and constraints:*

This step deletes those concepts not included in $G(CoI)$, which is obtained by the union of CoI concepts and all its parents. Pruning a concept implies the pruning of all generalization relationships in which that concept participates.

Similarly, we prune the irrelevant constraints. A constraint is irrelevant if not all the concepts that appear in it are included in $G(CoI)$. We call O_1 the ontology resulting from this step.

In figure 2, we have that *GeographicalAgent* and *University* are pruned because they are not contained in the set $G(CoI)$.

- *Pruning unnecessary Parents:*

Not all the concepts of O_1 are needed in the CS . The strictly needed concepts

(*NeededConcepts*) are given by the union of the concepts of direct interest and the concepts that appear in the definition of the remaining constraints. The other concepts that are parents of *NeededConcepts* and are not children of any concept in *NeededConcepts* are deleted. As a result we obtain O_2 .

In figure 2, the parents of *Agent* (*Thing* and *Individual*) are deleted, because they are parents of a needed concept (*Agent*), and none of its parents are needed concepts.

- *Pruning unnecessary generalization paths:*

O_2 may contain several duplicated specialization paths that can be deleted when their elements are not included into *NeededConcepts*, and, besides, do not participate in other specialization paths.

In figure 2 one specialization chain between *Agent* and *Person* is deleted (*Person* \rightarrow *Primate* \rightarrow *Mammal* \rightarrow *Animal* \rightarrow *Agent*), because none of its classes are needed elements and there is a shorter generalization path (*Person* \rightarrow *Legal Agent* \rightarrow *Social Being* \rightarrow *Agent*) between the same ends.

- *Pruning individuals:*

This step removes the instances of the ontology such that all their classifiers have been deleted in the previous steps. The result of this step is O_p .

In table 1 we may see the results of the application of our pruning activity in a case study [4].

Table 1. Results of the application of our method to a case study

Kind of element	O_x	O_p	After automatic refactoring	CS
Entity Types	2,268	200	44	31
Relationship Types	1,191	85	45	45

3.4 The state of the pruning activity

Until now, we have defined the pruning activity for UML [13], and we have also implemented a prototype that prunes UML ontologies. In addition, several case studies have been tried [4, 5] to validate the efficiency of the pruning activity. Furthermore, we adapted our method to prune OWL ontologies in [14]. We also defined in the same document a taxonomy that classifies the different ways of selecting the concepts of direct interest.

We think that our pruning the method can be also used in other contexts. In particular, we would like to use it in the development of domain ontologies, which is an ontology that represents the generic concepts related to a particular domain. We also would like to implement the pruning method in a CASE tool which provides independence of the selection strategy.

4 Refactoring Conceptual Schemas

Normally, a pruned ontology O_p cannot be accepted as a final conceptual schema because it tends to be quite large and can be greatly improved. 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.

4.1 Actual refactoring approaches

The refactoring concept appears with Opdyke in 1992 [15]. Nevertheless, the most accepted definition is the Fowler definition [16]. He defines a refactoring as *a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour*.

It is not until 2001 that Sunye et al. [17] define the refactoring process to conceptual schemas written in UML, defining a set of refactoring operations applicable to class diagrams. As far as we know, this is the first paper that shows the refactoring as an analysis and design technique instead of a technique that applies only to source code. Nevertheless, nowadays it does not exist a public repository of CS refactoring operations yet.

Nowadays, some of the efforts in this field are directed to automatically detect when a refactoring operation can be applicable (also known as detecting refactoring opportunities). The main works about detecting refactoring opportunities in CS are addressed by Van Gorp et al. [18]. They define a way to specify UML refactoring operations that allows automating its detection in several cases. Moreover, Porres defines in [19] a way to represent refactoring operations using transformation rules. This approach allows automating the refactoring detection in several cases, as it can be seen in the example used in his paper.

4.2 Our proposal

The objective of our refactoring activity is to obtain a CS such that:

- CS and O_p are functionally equivalent with respect to the IS requirements
- CS is obtained only by using refactoring operations
- CS is more compact than O_p , and it is closer to the structure that the designer has in mind

Although the refactoring is commonly performed by the designer, important parts of the activity can be automated when the IS requirements are formalized. Thus, our activity is composed by an automatic refactoring and a manual refactoring.

4.2.1 Automatic Refactoring

The main objective of the automatic refactoring is to automatically detect the maximum number of refactoring opportunities, and apply their associated refactoring operations.

Up to now, we have directed our efforts to detect the redundancies contained in O_p . We say that two entity or relationship types are redundant (or equivalent) in a conceptual schema if they have the same extension in all possible states of the corresponding information base [20]. For example, in Figure 2 *Legal Agent* and *Person* are redundant.

Until now, we have defined two similar but independent conditions for redundancy that detects when two generalizable elements are redundant:

- Single Child: There exists an abstract concept that has only one child.
- Single Parent: There exists an abstract concept that has only one parent.

Once detected, the redundancies are deleted using the *CollapseHierarchyDown* [16] operation. Furthermore, our conditions for redundancy require to know the value of the meta-attribute *isAbstract*. For the purposes of our redundancy conditions, we make the conservative assumption that a concept in a pruned ontology O_p is abstract if it is never referenced in the specification of the system operations.

4.2.2 Manual refactoring

Unfortunately, not all the refactoring opportunities can be detected automatically, because they depend on the designer criteria. Thus, a phase is necessary that allows the designer to execute refactoring operations on O_p to improve it.

4.3 The state of the refactoring activity

Until now, we detected and defined the *single child* and *single parent* redundancies in the UML case [3]. We have also validated this approach on several case studies [4, 5]. Table 1 shows the results of the application of this activity in [4].

We have more work to do in this phase than in the previous one, because we have not begun to work deeply in this activity. The further work of this activity is:

- Adapt the concept of conceptual schema refactoring to ontology refactoring.
- Formally define the refactoring opportunities of the automatic refactoring.
- Define a repository of refactoring operations, applicable both to CS and ontologies, taking into account their special constructs, such as instances, metaclasses, association classes, and so on.
- Automate the refactoring activity as possible. To achieve this objective we will work both in the detection of new refactoring opportunities, finding out more redundancy conditions and improving the automatic identification of abstract elements

At the end of this work, we would like to implement the refactoring activity in a CASE tool.

5 Contributions of this work

We believe that our whole approach may yield important benefits in the assisted creation of conceptual schemas. Its advantages are that it allows reusing the knowledge written by domain experts in a semiautomatic way. It is definitely directed to IS designers, so it will be implemented in a CASE tool, using UML as the ontology language. Although large ontologies may be used to learn about a domain, they are usually too large to do it possible. We think the combination of the pruning and refactoring activities might be used to obtain a focused ontology that allows learning knowledge more easily from large ontologies.

However, we think our individual activities may also yield important benefits in the pruning and refactoring of conceptual schemas. In particular, our pruning activity is the only UML-based method, and, as far as we know, it is actually the method that

generates the smallest pruned ontologies. On the other hand, the rest of pruning methods are strongly influenced by their selection criteria, and must be redefined if the selection criteria changes. Our pruning method is independent of both the selection criteria and the ontology language [14].

The refactoring activity may also be relevant to actual research, because we would like to define both the refactoring for ontologies in a formal way and a repository of ontology refactoring operations. Furthermore, the refactoring opportunities identified in this thesis might be used by other people to improve automatically their conceptual schemata.

References

1. Gruber, T.R., *Toward Principles for the Design of Ontologies for Knowledge Sharing*, in *International Journal of Human and Computer Studies*. 1995. p. 907 - 928.
2. Mili, H., F. Mili, and A. Mili, *Reusing Software: Issues and Research Directions*. TSE, 1995. **21**(6): p. 528-562.
3. Conesa, J., X.d. Palol, and A. Olivé, *Building Conceptual Schemas by Refining General Ontologies*, in *DEXA'03*. 2003, Springer. p. 693 - 702.
4. Conesa, J. and X.d. Palol, *A Case Study on Building Conceptual Schemas by Refining General Ontologies*. 2003, Report number LSI-03-17-R. LSI UPC.
5. Conesa, J., *A Case Study on Pruning General Ontologies for the Development of Conceptual Schemas*. 2004, Report number LSI-04-18-R. LSI UPC: Barcelona. p. 22.
6. Swartout, B., et al., *Toward Distributed use of Large-Scale Ontologies*, in *Proc. 10th. Knowledge Acquisition for Knowledge-Based Systems Workshop, Canada*. 1996.
7. Peterson, B.J., W.A. Andersen, and J. Engel, *Knowledge Bus: Generating Application-focused Databases from Large Ontologies*, in *KRDB'98*. 1998. p. 2.1-2.10.
8. Kietz, J.-U., A. Maedche, and R. Volz, *A Method for Semi-Automatic Ontology Acquisition from a Corporate Intranet*, in *Proceedings of EKAW-2000 Workshop*. 2000.
9. Wouters, C., et al., *A Practical Walkthrough of the Ontology Derivation Rules*, in *DEXA'02*. 2002, Springer. p. 259-268.
10. Yamaguchi, T., *Constructing Domain Ontologies Based on Concept Drift Analysis*, in *IJCAI-99. Workshop on Ontologies and Problem-Solving Methods*. 1999.
11. Navigli, R. *Automatically Extending, Pruning and Trimming General Purpose Ontologies*. in *International Conference on Systems, Man and Cybernetics*. 2002. Tunisia.
12. Larman, C., *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design*. 1998: Prentice Hall.
13. Conesa, J. and A. Olivé. *Pruning Ontologies in the Development of Conceptual Schemas of Information Systems*. in *ER2004*. 2004. Shangai, China: Springer.
14. Conesa, J. and A. Olivé. *A General Method for Pruning OWL Ontologies*. in *ODBASE'04*. 2004. Larnaca, Cyprus: Springer.
15. Opdyke, W.F., *Refactoring Object-Oriented Frameworks*. *Ph.D. Thesis*. 1992, University of Illinois.
16. Fowler, M., *Refactoring: Improving the Design of Existing Code*. 1999: Addison-Wesley.
17. Sunye, G., et al., *Refactoring UML Models*, in *UML 2001*, M. Gogolla and C. Kobryn, Editors. 2001, Springer. p. 134 -148.
18. Gorp, P.V., et al., *Towards Automating Source-Consistent UML Refactorings*, in *UML 2003*. 2003, Springer. p. 144 - 158.
19. Porres, I., *Model Refactorings as Rule-Based Update Transformations*, in *UML 2003*. 2003, Springer. p. 159 - 174.
20. Calvanese, D.L., M.; Nardi, D., *Description Logics for Conceptual Data Modeling*, in *Logics for Databases and Information Systems*. 1998, Kluwer. p. 229-263.