Deriving Binary Relation Types From Concept Types

Philippe A. Martin [1] and Jérémy Bénard [2]

[1] EA2525 LIM, ESIROI I.T., University of La Réunion, F-97490 Sainte Clotilde, France
+ adjunct researcher of the School of I.C.T. at Griffith University, Australia
Philippe.Martin@univ-reunion.fr
[2] GTH, Logicells, 55 rue Labourdonnais, 97400 Saint-Denis, France
Jeremy.Benard@logicells.com

Abstract. This article describes an ontology design pattern. It helps normalizing knowledge, reducing the introduction of new relation types and keeping all the types organized. Thus, it leads knowledge providers to represent knowledge in more normalized, precise and inter-related ways, hence in ways that help the matching and exploitation of knowledge from different sources. This pattern is also a knowledge sharing best practice that is domain and language independent. It can be used as a criteria for measuring the quality of an ontology.

Keywords: Knowledge sharing, best practices, relation type generation

1 Introduction

Ontology Design Patterns (ODPs) are "modeling solutions to solve a recurrent ontology design problem" [1]. Many ODPs have been found, e.g., about 160 are currently registered in the "ODP catalog at http://ontologydesignpatterns.org". However, the thousands of ontologies (UML schemas included) that exist on the Web are poorly inter-connected and heterogeneous in their design. It is then difficult for automated agents to *compare or match* such independently created knowledge representations (KRs, e.g., types or statements) to know if some KRs are equivalent to others or specializations of others. Thus, it is difficult for people and automated agents to *align and aggregate* – and, thus relate, infer from, search or exploit – KRs or ontologies.

In other words, there is a need for ODPs specifically aimed for knowledge sharing and, more precisely, for solving *the problem of leading knowledge providers to create more matchable and re-usable KRs*. As later detailed, this implies leading them to create more precise, normalized, well related and easy-to-understand KRs. In order to be adopted, these ODPs should also be easy to follow and easy to use as criteria for automatically measuring the quality of an ontology, to help developing an ontology or selecting ontologies to re-use. Finally, the ODPs – or, at least the knowledge sharing ODPs – should be well inter-related by semantic relations to help people i) know about them and their advantages, and ii) select those they want to commit to. Then, tools can check or enforce these commitments.

This article proposes such a knowledge sharing focused ODP which is also a best practice (BP). This BP, which in this article will now be referred to as ABP, is: *"using binary relation types directly derived from concept types"*. No ODP catalog appears to include similar ODPs. Like most BPs, it is domain and language independent and it can be used for any dataset. The interest of using binary relation types only for KR is well understood, if only because many KRLs (KR languages) only support binary relations. Hence, this article does not explain this interest.

2 Deriving Relation Types from Concept Types

In this article, types that are not relation types (RTs) are refered to as "concept types" (CTs; "classes" in RDF and OWL). A relation is not a type. Since ABP is language independent, this article uses a general terminology. It is compatible with those for Conceptual Graphs, RIF-FLD [2] and the W3C Framework for Logic Dialects of the Rule Interchange Format.

When buildings RCs, it is better to use RTs with definitions than without. It makes some information explicit and ensures that every distinction in the (subtype) hierarchy of RTs is also included in the CT hierarchy. This last point is important for two reasons. First, it avoids that some knowledge providers develop distinctions only in the RT hierarchy while others develop distinctions only in the CT hierarchy, thus leading to *(automatically) undetected redundancies* within a shared knowledge base or in different ontologies. Second, it ensures that any distinction can be used – *without loosing in knowledge representation and matching possibilities* – with both its CT form and its RT form. More possibilities come from the CT form since i) unlike RTs, CTs can be quantified in many different ways (e.g., "3 landings", "all landings" or "8% of landings" can only be described via the CT "Landing", not the RT "r__landing"), ii) it is easier to organize CTs (by subtype relations) than RTs, and iii) the number of used or re-usable existing CTs is much greater than the number of used or re-usable RTs.

These advantages of using *defined RTs* come for free when the RTs are automatically derived from CTs and hence defined with respect to them. *Furthermore*, such derivations permits a system to display fewer types in the RT hierarchy (which is then easier to read and grasp). Indeed, the derived RTs may be left hidden or may not have to be created all. This last option was used in the knowledge server Ontoseek [3] and is used in the knowledge base server WebKB (www.webkb.org; [4]). In Ontoseek, any type derived from the noun-related part of the lexical ontology Sensus could be re-used as a CT or a RT. WebKB also re-uses a lexical ontology derived from WordNet – the "Multi-Source Ontology" (MSO [5]). However, unlike Ontoseek, WebKB only allows the subtypes of certain types to be re-used as RTs. This is defined by specifications that users can adapt. More precisely, this is defined by relation signatures which are directly associated to certain top-level CTs. The MSO includes more than 75,000 categories (mainly types) and relates them by more than 100,000 relations.

Table 1 illustrates the approach and then gives rules that would actually generate the derived RTs. These rules permit to formalize the framework. They rely on the functions

Table 1. RIF-FLD PS rules for automatically deriving a binary RT from a CT (and, if needed, doing so for all its subtypes) based on a kind of signature associated to this CT. In these examples, the types created by the authors of this article have no prefix to indicate their namespace. RIF-FLD PS, the Presentation Syntax of RIF-FLD, is used because it is both expressive and rather intuitive. RT names begin by "r__" and function names begin by "f__". Logical rules are used since RIF-FLD is used but here logical equivalences could also be used. ":-" means "<=".

The derived RT does not have to be explicitly defined. Its signature is directly associated to the CT via a relation of type r__signature_for_derived_binary_relation or a function of type f__derived_binary_relation. Thanks to their definitions, the derived RT is automatically created (see the next paragraph in bold characters). A CT may have different RT signatures associated to it, as long as the signatures are "un-comparable" (i.e., as long as none specializes another).

```
r__signature_for_derived_binary_relation ( Father   List ( Animal  Male )  )
     //-> derives the RT  r__father  that has for domain an Animal and range a Male

Forall  ?t  (  r__signature_for_derived_binary_relation ( ?t   List ( Thing  ?t ) )
                :-  ?t  ##  thing_usable_for_deriving_a_binary_relation_with_it_as_destination
    //"?st  ## ?t" <=> subtypeOf (?st ?t); this rule derives the expected RT for each subtype of
    //      Thing_usable_for_deriving_a_binary_relation_with_it_as_destination

Forall  ?t   Exists ?r
  And (  ?r = f__ derived_binary_relation ( ?t   List ( Agent  Object ) )
         Forall  ?agent  ?object    And ( r__agent (?t  ?agent)   r__ object (?process  ?object)
                                          )  :-  ?r (?agent  ?object)
      )  :-   ?t  ##  Process       //-> derives the expected RT for each subtype of Process
```

Furthermore, the derived RTs have the same subtype relations as the CTs they derive from. However, to keep things simple, it is here assumed that no RT with the same name as the derived RT has previously been manually created. The RT name is created by taking the CT name, lowering its initial and prefixing it with "r__". The functions f__denotation_of_type_name, f__type_name, f__cons, f__cdr, f__lowercase used below are identical to their counterparts (without the prefix "f__") in KIF.

```
Forall  ?t   ?r__t    ?t_domain   ?t_range
         ?t_supertype    ?r__t_supertype    ?t_sup_domain    ?t_sup_range  (
   And (  rdfs:domain (?r__t   ?t_domain )       rdfs:range (?r__t    ?t_range )
          ?r__t  =  f__denotation_of_type_name
                     ( f__cons ( f__lowercase ( f__car ( f__type_name ( ?t ) ) ) )
                       f__cdr ( f__name ( ?t ) )    )
          ?r__t    #   ?r__t_supertype
              :-  And (  ?t   #  ?t_supertype
                         ?r__t_supertype  =  f__ derived_binary_relation
                                              ( ?t_supertype
                                                List ( ?t_sup_domain  ?t_sup_range )  )    )   )
   :-   ?r__t    =    f__ derived_binary_relation ( ?t   List ( ?t_domain  ?t_range ) )   )

Forall  ?t   ?t_domain   ?t_range (
   Exists  ?r__t  ( ?r__t    =    f__ derived_binary_relation ( ?t    List ( ?t_domain  ?t_range ) ) )
   :-  f__signature_for_derived_binary_relation ( ?t   List ( ?t_domain  ?t_range ) )  )
```

f__type_name and *f__denotation_of_type_name* which are identical to the KIF functions *name* and *denotation* formalized in the documentation of KIF [6]. In WebKB, no such rules are executed (a more efficient and ad-hoc process is used): during the analysis of RCs, when a CT is used in places where RTs are expected, WebKB simply checks that one of the signatures associated to the CT is respected and acts as if the relevant derived RT was actually used. Thus, in WebKB, there is no need to use the actual names of the virtually derived RTs: the CT names can be used directly. As described by Table 1, signatures are inherited along subtype relations between CTs and an error is generated if a CT is associated to two signatures that are "comparable". This approach and ODP seem original.

3 Acknowledgment

We thank one of our reviewers for its very positive review on our initially submitted article.

4 References

1. Presutti, V., Gangemi, V.: Content Ontology Design Patterns as Practical Building Blocks for Web Ontologies. In: ER 2008, Spaccapietra S. et al. (eds.)
2. Boley, H., Kifer, M. (eds.): RIF Framework for Logic Dialects (2nd edition). W3C Recommendation, http://www.w3.org/TR/2013/REC-rif-fld-20130205/ (2013)
3. Guarino, N., Masolo, C., Vetere, G.: Ontoseek: Content-based Access to the Web. IEEE Intelligent Systems, vol. 14, No. 3, pp. 70–80 (1999)
4. Martin, Ph.: Collaborative knowledge sharing and editing, IJCSIS, vol. 6, Issue 1, pp. 14–29 (2011)
5. Martin, Ph.: Correction and Extension of WordNet 1.7. In: LNAI 2746, pp. 160–173. See also http://www.webkb.org/doc/MSO.html
6. Genesereth, M., Fikes R.: Knowledge Interchange Format, Version 3.0, Reference Manual. Technical Report, Logic-92-1, Stanford Uni., http://www.cs.umbc.edu/kse/ (1992)