Directly deriving binary relation types from concept types, especially process or role types

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

Philippe Martin[a] and Jérémy Bénard [b]
[a] *Université de La Réunion (and adjunct researcher of Griffith University, Australia)*
EA2525 LIM, Saint-Denis de la Réunion, F-97490, France. E-mail: Philippe.Martin@univ-reunion.fr
[b] *Logicells, 3, rue Désiré Barquisseau, 97410 Saint-Pierre, France. E-mail: jeremy.benard@logicells.com*

Abstract. This article proposes an ontology design pattern for leading knowledge providers to represent knowledge in more normalized, precise and inter-related ways, hence in ways that help the matching and exploitation of (rather independently created) knowledge. This pattern is a knowledge sharing best practice that is domain and language independent and it can be used as a criteria for measuring the quality of an ontology. This pattern is: "using binary relation types directly derived from concept types, especially role types or types of process with nominal expressions as names". The article explains and illustrates this pattern, and relates it to other patterns and general ontology quality criteria. It also provides an ontology for automatically deriving relation types from concept types (e.g., those from lexical ontologies such as those derived from the WordNet lexical database). This derivation helps normalizing knowledge, reduces having to introduce new relation types and helps keeping all the types organized.

Keywords: knowledge sharing, knowledge normalization, knowledge matching, best practices, relation type generation

1. Introduction

Ontology Design Patterns (ODPs) are "modeling solutions to solve a recurrent ontology design problem" [16]. Many ODPs have been found, e.g., 160 are currently registered in the "ODP catalog at http://ontologydesignpatterns.org" which, in this article, will now be referred to as "ODPC". However, the thousands of ontologies (UML schemas included) that have been proposed so far are still poorly inter-connected and heterogeneous in their design. It is then difficult for people and automated agents to *compare or match* 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 search, align, aggregate – and, more generally, relate, infer from 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 and best practice (BP) and relates it to other ones, via specialization relations and "gradual pattern" relations. This advocated BP, which in this article will now be referred to as ABP, is: *"using binary relation types directly derived from concept types, especially role types or types of process with singular nominal expressions as names"*. No ODP catalog appears to include ODPs similar to this one or

to any of its parts. Like most BPs, it is domain and language independent and it can be used for any dataset. The sections 2, 3, 4 and 5 explain, formalize and illustrate the different parts of ABP. Section 6 relates them to other ODPs and thereby also give more rationale.

2. Using binary relations

ABP starts by advocating the use of "*binary relations*" (or "properties" in RDF and OWL). Types that are not relation types are "concept types" ("classes" in RDF and OWL). However, from now on, to avoid a too frequent use of the word "type", it is sometimes left implicit when referring to a "concept/relation type" or to "a type and/or any of its instances". The word "individual" will be added when a type is *not* referred to.

Since ABP is language independent, this article uses a general terminology, one compatible with RIF-FLD, the W3C Framework for Logic Dialects of the Rule Interchange Format [17]. For its formal textual examples, this article uses RIF-FLD PS, the Presentation Syntax of RIF-FLD. Indeed, this notation is both expressive and rather intuitive. In the examples of this article, relation names begin by "r__" and function names begin by "f__". Logical rules are used since RIF-FLD is used and since this shows the direction the implications are expected to be used. However, in each case, a logical equivalence could also be used instead.

Following ABP does not prevent using non-binary relations as long as definitions or rules are also provided to enable the automatic translation of "KRs using non-binary relations" into "KRs using binary relations". Table 1 illustrates such rules for various kinds of use cases (only the third row is also about the focus of Section 3: deriving a relation from a concept).

One reason why such rules are useful for knowledge sharing is that binary relations can be "compared" while relations of different arities generally cannot be (two types or KRs are "comparable" if and only if an equivalence or specialization relation between them has been directly stated or can be inferred). Thus, KRs using binary relations can be ordered by generalization relations, typically, implications. This is more difficult with KRs using relations of different arities, thus reducing possibilities for knowledge matching or inferences. E.g., as illustrated by Table 1, many "directly un-comparable" relations of different arities can be translated into binary relations of type r__list_of_surrounding_entities (which uses a list). Then, they can be "compared".

A related reason is that they make more information explicit. Normalizing, precising and supporting

Table 1

Examples of how and why defining a given relation type with respect to other types (notes: here, rules are used for the definitions; the RIF-FLD PS notation is used in the non-highlighted parts; variables begin by "?"; "S1:- S2" can be read "If S2 then S1")

If you wish to (re-)use non-binary relations, as in

```
r__spatial_entity_between_3_other_ones ( Jack  Joe John  Mary )
Exists ?X  r__spatial_entity_between_2_other_ones (?X Joe John)
```

, instead of using binary relations, as in

```
r__list_of_surrounding_entities ( Jack  List( Joe  John  Mary ) )
Exists ?X  r__list_of_surrounding_entities ( ?X  List( Joe  John ) )
```

, then provide ways to translate the 1st ones into the 2nd ones, e.g.,

```
Forall  ?A  ?B  ?C  ?D
   r__list_of_surrounding_entities ( ?A  List( ?B  ?C  ?D ) )
     :- r__spatial_entity_between_3_other_ones ( ?A  ?B  ?C  ?D )
```

, since it is then much easier to make inferences, e.g., ?X = Jack and the above 3rd statement specializes (hence implies) the 4th

The above approach also works for contextualizations, e.g.,

```
r__list-of-surrounding-entities_at-time ( Jack  Joe  John  D-Day )
```

can automatically be translated into the binary relation

```
r__list_of_surrounding-entities ( Jack_at_D-Day
            List (Joe_at_D-Day  John_at_D-Day) )
```

This cannot be specified in RIF PS but something similar can be:

```
Forall  ?A  ?B  ?C  ?time_T
  Exists ( ?A_at_time_T  ?B_at_time_T  ?C_at_time_T )
    And ( r__list_of_surrounding_entities (?A_at_time_T
                  List (?B_at_time_T  ?C_at_time_T) )
          r__extended_specialization (?A  ?A_at_time_T)
            r__time ( ?A_at_time_T  ?time_T )
          r__extended_specialization (?B  ?B_at_time_T)
            r__time ( ?B_at_time_T  ?time_T ) )
    :- r__list-of-surrounding-entities_at-time (?A  ?B  ?C  ?time_T)
```

Similarly, if you wish to use relations for processes, as in

```
r__landing ( Joe  Omaha_Beach  D-Day )
r__defining (Joe  Square)
```

, instead of using classic primitive binary relations, as in

```
Exists  ?landing
  And (  ?landing  #  landing     // "?i  #  ?t" <=> instanceOf (?i ?t)
         r__agent(?landing Joe)  r__place(?landing Omaha_Beach)
         r__time(?landing  D-Day)  ) )
Exists  ?defining
  And (  ?defining  #  defining        r__agent (? defining  Joe)
         r__object (?defining  "square")  ) )
```

, then provide ways to translate the 1st ones into the 2nd ones, e.g.,

```
r__directly_derived_relation ( Landing  r__landing )
r__directly_derived_relation ( Defining  r__defining )

Forall  ?rel  ?process  ?agent  ?time  ?place
   And (  r__agent (?process  ?agent)   r__place (?process  ?place)
          r__time (? process  ?time)  )
     :- And (  ?rel ( ?agent  ?place  ?time )
               r__process ( ?rel  ?process )  )

Forall  ?rel  ?process  ?agent  ?object
   And (  r__agent (?process  ?agent)  r__object (?process  ?object) )
     :- And (  ?rel ( ?agent  ?object )
               r__directly_derived_ relation ( ?process  ?rel )  )
```

, since it is then much easier to make inferences,
e.g., for the statement in the next line, a match for ?X is Joe

```
Exists ?A  And( r__agent (Landing  ?A)  r__agent (Defining  ?A) )
```

knowledge comparability have strong relationships. They are represented in Section 6.

In practice, with a KR language (KRL) allowing "contexts" and sets or lists, it is easy to avoid the use of relations with arity greater than 2. A "context" (or "contextualizing statement") is a meta-statement specifying restrictive conditions for the contextualized statement to be true, e.g., via temporal relations or modalities. Although RIF-FLD PS is not restricted to first-order logic, it lacks a construct for expressing contextualizations in simple ways, as in KIF [9] for example. However, the second row of Table 1 shows how simple contextualizations can still be represented - albeit in a rather cumbersome way - using binary relations. To that end, this example uses an adaptation of the ODP named "Context Slices" in ODPC [20]. It relies on introducing "concept individuals within a context"and relating them to their context as well as to their context-independent counterpart. This is an alternative to the more common approach of reifying a statement and asserting a relation individual between the reification and the context. With the reification based approach, handling contexts is a bit more difficult when simple KR management tools are re-used and extended. Both approaches lead to rather lengthy statements and are ad-hoc since they require extensions to inference engines to fully handle them correctly. Therefore, *from a knowledge modeling and sharing viewpoint, a BP is to i) use a KRL that handles contexts (or else design and use equivalent ad-hoc concise constructs), and then ii) provide or use rules for translating into the various ways to represent contexts in other KRLs. The same idea applies for the many ODPs dealing with the problems of translating "KRs using high expressive constructs" into "KRs using lower expressive constructs"* (e.g., in ODPC, there are many ODP for translations into OWL or from OWL).

It should also be noted that the practical absence of "*necessity* to use non-binary relations" is compatible with formal proofs that "besides unary and binary relations, there must *exist* at least one ternary relation in order to *generate* all possible relations" [5].

There is no claim here that the idea of "translating non-binary relation types into binary ones or directly using them" is original. Yet, it should be an ODP for various reasons: i) it is useful, ii) some claims seemingly about the necessity of using non-binary relations are actually claims about the need for constructs supporting different kinds of contexts (e.g., [21]), iii) this best practice is often ignored by – or unknown to – users of KRLs allowing non-binary relations.

3. Deriving relation types from concept types

ABP advocates the use of - or specifications of translations into - binary relation types "*directly derived from concept types*". This means that each of these relations is defined with respect to one and only one concept. In other words, if the relation is defined via a genuine "type definition" instead of a rule, apart from the concepts in the signature of the relation, there is only one concept in the definition. The word "directly" refers to this "only one" restriction. A concept type may have multiple "directly derived relation types" if they have "un-comparable" signatures (i.e., if none specializes another one). The third row of Table 1 illustrates a way to directly derive a (binary or not) relation from a concept without using a genuine definition but using a rule and the relation individual of type r__directly_derived_relation. The first two rows illustrate the definitions of non-binary relations mainly with respect to binary relations. This is useful as a intermediary step: the final step – deriving these last binary relations from a concept (e.g., named "List_of_surrounding_entities") – is not illustrated in Table 1.

When genuine definitions or rules are manually given for each derived relation, as illustrated in the third row of Table 1, the advantages of the approach (over directly using relations without defining them) only come from the existence of this definition. It makes some information explicit and ensures that every distinction in the (specialization) hierarchy of relations is also included in the concept hierarchy. This last point is important for two reasons. First, it avoids that some knowledge providers develop distinctions only in the relation hierarchy while others develop distinctions only in the concept 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 knowledge representation and matching possibilities* – with both its concept form and its relation form. More possibilities come from the concept form since i) unlike relations, concepts can be quantified in many different ways (e.g., "3 landings", "all landings" or "8% of landings" can only be described via the concept "Landing", not the relation "r__landing"), ii) it is easier to organize concepts (by subtype relations) than relations, and iii) the number of used or re-usable existing concept types is much greater than the number of used or re-usable relation types.

These advantages come for free when the relation types are automatically derived from concepts. Fur-

Table 2

Rules for automatically deriving a binary relation type from a concept type (and, if needed,doing so for all its subtypes) based on a kind of signature associated to this concept type (note: in these examples, the types created by the authors of this article have no prefix to indicate their namespace).

Table 1 gave examples of how a rule can define a relation type with respect to a concept type. This had to be done for each relation type. Here, the approach is simpler. The derived relation type does not have to be explicitly defined. Its signature is directly associated to the concept type 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 relation type is automatically created (see the next paragraph in bold characters). A concept type may have different relation types signature associated to it, as long as the signature are "un-comparable" (i.e., as long as one does not specialize another).

```
r__signature_for_derived_binary_relation ( Father  List ( Animal  Male ) )
    //-> derives a relation type  r__father  that has for domain an Animal and range a Male  (note: this line is a comment)

Forall  ?t  r__signature_for_derived_binary_relation ( ?t  List ( Thing  ?t ) )
            :-  ?t  #  thing_usable_for_deriving_a_binary_relation_with_it_as_destination
    //-> derives the expected relation type 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 relation type for each subtype of Process
```

Here are rules that permit such derivations . Furthermore, the derived relation types have the same subtype relations as the concept types they derive from. However, to keep things simple, it is here assumed that no relation with the same name as the derived relation has previously been manually created. The relation type name is created by taking the concept type 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 ) )
```

Other rules can be built upon these last ones, e.g., this rule for deriving functional binary relations :

```
Forall  ?t  ?t_domain  ?t_range  Exists  ?r__t  And ( ?r__t  =  f__ derived_binary_relation ( ?t  List ( ?t_domain  ?t_range ) )
                                                      ?r__t  ##  owl:FunctionalProperty  )  //"##" means "is instance of"
   :-  f__signature_for_derived_functional_binary_relation ( ?t  List ( ?t_domain  ?t_range ) )
```

thermore, doing so permits a system to i) hide the automatically derived relation types in the relation type hierarchy (which is then easier to read and grasp), or ii) not actually create them at all. This second option was used in the knowledge server Ontoseek [10] and is used in the shared/personal knowledge base server WebKB (www.webkb.org; [11]). In Ontoseek, any type derived from noun-related part of the lexical ontology Sensus could be re-used as a concept type or a relation type. WebKB also re-uses a lexical ontology derived from WordNet but only allows the subtypes of certain types to be re-used as relation types and this is defined by specifications which users can adapt. More precisely, this is defined by relation signatures which are directly associated to certain top-level concept types. Table 2 illustrates the approach and then gives rules that would actually generate the derived relation types (the next section complements this framework by giving an ontology of the concept types these rules can be applied to). These rules permit to give a formalization of the framework. They rely on the functions *f__type_name* and *f__denota-*

tion_of_type_name which are identical to the KIF functions *name* and *denotation* formalized in the documentation of KIF [9]. In WebKB, no such rules are executed: when a concept type is used in places where relation types are expected, WebKB simply checks that one of the signatures associated to the concept type is respected and acts as if the relevant derived type was actually used. Thus, in WebKB, there is no need to use the actual names of the virtually derived relation types: the concept type names can be used directly. As in the framework described by Table 2, signatures are inherited along subtype relations between concept types and an error is generated if a concept type is associated to two signatures that are "comparable". This approach and ODP seem original.

4. Deriving from role types and processes

ABP advocates the derivation of concept types, "*especially role types or types of process*". The third row of Table 1 illustrated this for processes. In this article, "process" refers to a "situation" (something that *occurs* in a real/imaginary region of time and space) that is not a "state", and hence that makes a change. These conceptual distinctions come from the Situation Semantics [2] and are the basis of John Sowa's first top-level ontology [18]. There are reused in this article for at least the following reasons:

- They are rather intuitive.
- They generalize other well known types, e.g., Perdurant from Dolce [3] is subtype of Process.
- They are very adequate for the signatures of thematic relations [4], e.g., r__agent, r__recipient, r__cause, r__instrument. Such types can actually be seen as particular top-level types of relations from a process.
- In this article, a "role (type)" (e.g., Agent, Experiencer, Recipient, Cause, Instrument) is a concept which is defined – or could be defined – as being the destination of a thematic relation. This informal definition of a role is a bit more general than what is usually thought to be a role [13] but here it is sufficient: process and role types (as defined here) can be used for deriving concept types into binary relation types.
- Thematic relations or their subtypes can also be used for defining most relations. Thus, doing so normalizes KRs.
- Most statements implicitly or explicitly refer to a process. Representing it, either directly (and then using thematic relations or subtypes of them) or via relations directly derived from a process, strongly normalizes KRs. Not doing so, which unfortunately is the case in most ontologies, amounts to loosing precisions and a lot of KR comparison possibilities.

Table 3 displays subtype relations between types related to *the type of "things usable for directly deriving a binary relation"*. Only its subtypes can be used for deriving a binary relation; this includes process types and roles. Table 4 displays common top-level types of relations from a process, most of which are thematic relations. Table 4 re-uses top-level types shown in Table 3. All the types in these tables are part of the "Multi-Source Ontology" (MSO [12]) which is accessible and cooperatively updatable via WebKB. Hence, the names in these tables are names accessible via this server. However, these tables have not previously been published.

The MSO includes more than 75,000 categories (mainly types) and relates them by more than 100,000 relations. It categorizes WordNet types (and their instances) as well as types from various top-level ontologies (DOLCE included) with respect to the types shown in Table 3 or specializations of them. More precisely, about a hundred of top-level WordNet types and some more specialized WordNet types were manually set as subtypes of those in Table 3 or specializations of them. Thus, in the subtype hierarchy of the MSO for "things usable for directly deriving a binary relation", there are currently a bit more than 4800 process types, 2900 role types ("things playing some role"), 650 types of "attributes or qualities or measures" and 240 types of "description content/medium/container". This makes more than 8600 types usable for creating relation individuals without having to declare new relation types. The 4800 process types can also be used directly with "relations from a process". Finally, the types shown in Table 4 for these relations can (implicitly or explicitly) have for subtypes types derived from the 2900 role types. To sum up, the proposed approach and the MSO permit people and automated agents to create KRs that are well normalized, inter-related and comparable. Furthermore re-using the approach and content of the MSO to extend other ontologies is eased by the fact that i) the MSO relates, generalizes and specializes types from various other ontologies, and ii) it can be complemented online via WebKB.

In Table 3, the types named Relative_thing and Mediating_thing come from John Sowa's second top-level ontology [19].

Table 3

Some types and subtypes/supertypes for types of "thing usable for directly deriving a binary relation" (1st row) and their derived relation type (2nd row) (notes: disjoint types and unions of disjoint types are made explicit via UML-like annotations; for namespaces, XMLshortcuts are used but types created by the first author of this article have no prefix; "wn" refers to WordNet; "(*)" is the relation type signature for any set of arguments)

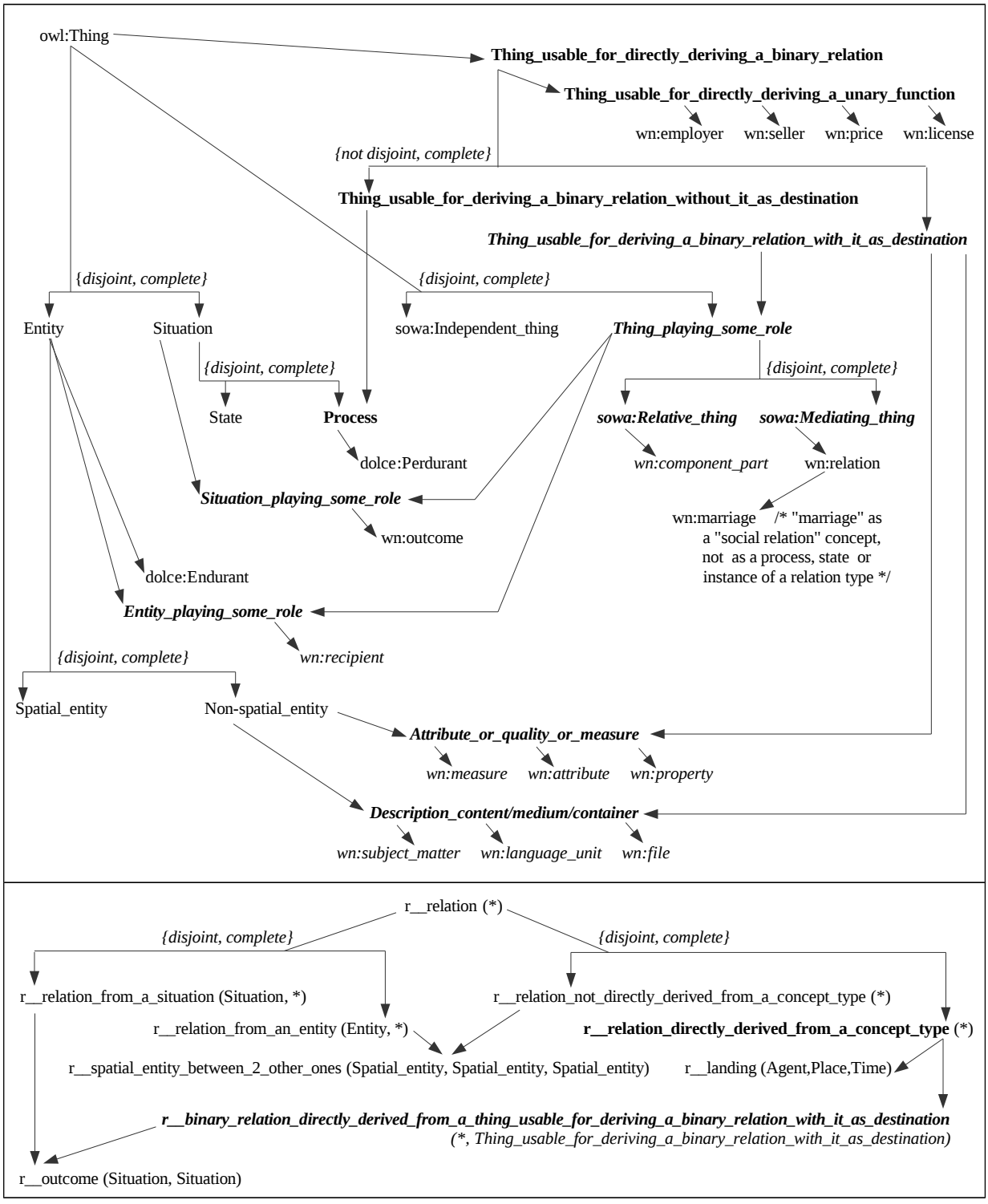
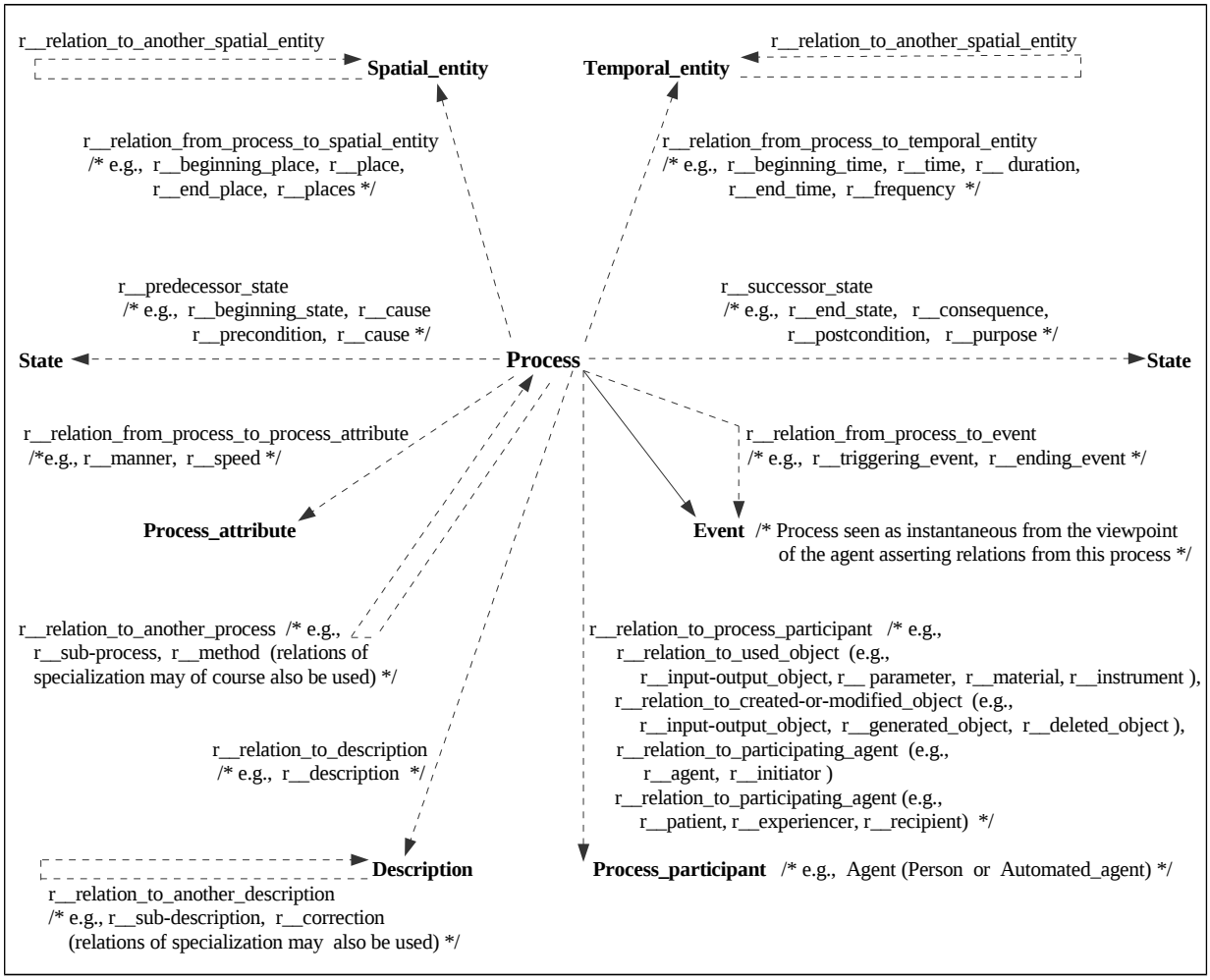


Table 4

Examples of common top-level types of relations from a process, most of which are thematic relations (notes: arrows with continuous lines are subtype relations; arrows with dashed lines are relations like UML associations, i.e., the source type is universally quantified and a cardinality/multiplicity is associated to the destination type; here, each cardinality/multiplicity is either "0 to many" or "1 to many", and is left implicit; comments are enclosed within "/*" and "*/"; "e.g.," is used for introducing subtypes)



To show how rules can be used to associate a signature to a concept type and thereby to a derived relation type, examples in Table 2 used a process type and the type of "things usable for deriving a binary relation with it as destination". Similar rules can be used for other types of "things usable for deriving a binary relation". The second row of Table 3 shows how the various relations types – derived or not from concept types – can be related by subtype relations. Organizing relations of different arities is permitted by the use of "*" in the relation signatures: it refers to any number of arguments. In Table 3, a signature is shown as an ordered list of comma-separated arguments, within parenthesis. Both KIF and RIF-FLD allow relations with a variable number of arguments. However, unlike in KIF, there is no special construct in RIF-FLD PS for definitions, hence for signatures.

ODPC includes the DOLCE+DnS-Ultralite ontology [7] as "content ODP" as well as smaller "content ODPs" extracted from it, e.g., "ActingFor" and "AgentRole". Its DnS (Descriptions and Situations) part includes some types which can be seen as subtypes of those in Table 4. It proposes many relations types which could be – but, it seems, are not – derived from process types, e.g., relation types with names such as "actsFor", "conceptualizes" or "defines". Yet,

some of its concept types have been aligned with OntoWordNet [8]. Thus, the ontology and approach proposed in this section (and the previous one) could be used to extend DOLCE+DnS-Ultralite. This would support more KR comparison possibilities.

5. Using singular nominal expressions for types

ABP advocates the use of "*role types or types of process with singular nominal expressions as names"*. Thus, this part of ABP advocates a lexical normalization. There is no required effort regarding role types since apparently they always have singular nominal expressions as names. More generally, this section advocates using "*singular nominal expressions for the names of all concept/relation types".* This generalization was not included in ABP to keep it simple: ABP is focused on concept type derivations (hence the title of this article). In ODPC this generalization would be in the "Naming ODP" category (which is currently empty).

First, here are inter-related reasons for using "relation types with nominal expressions as names":

- This eases the reading of relations via the *graph-oriented reading convention* according to which "X R: Y" is read "X has for R Y" or "Y is the R of X". E.g., "a landing has for r__agent Joe" or "Joe is the r__agent of a landing". All thematic relations and all the (derived or not) relation types in this article follow this convention. It is often given in the documentations of many frame-based or graph-based KRLs. Yet, when using these KRLs, people often add the prefix "has" to their relation types, as in "hasAgent". This is not a genuine problem since, when needed, such prefixes can be automatically removed, e.g., in order to have natural looking KRs when controlled natural languages are used. On the other hand, relations with suffix "Of" do not respect this convention. Finally, as is the case in KIF, functions are often formalized via functional relations that have the function result as last argument. For unary functions, this leads to the above cited convention.
- Assuming that in current ontologies, there are more types following the above convention that not, following it is a good strategy for increasing comparability between existing types. A better strategy is to create relation types following the convention and automatically derive their inverse relation types, e.g., via rules similar to those described in Section 3. These (implicitly or explicitly) derived relation types need not be displayed in the relation type hierarchy. Indeed, having all the displayed types following the same convention helps people understand the relation type hierarchy, complement it and correctly use its types (derived or not). This is unfortunately often not the case in ontologies re-using other ones, e.g., SUMO [14]. In the MSO this problem is avoided by always creating an inverse relation type when a re-used relation type does not follow the convention.
- This prevents the creation of "relation types with verbs as names". These types – and relation individuals using them – cannot be read with the *graph-oriented reading convention.* Thus, they may be misunderstood and incorrectly subtyped. People creating such types also tend not to give them definitions with respect to a process.

Here are inter-related reasons for using "concept types with nominal expressions as names":

- This prevents the creation of "concept types with verbs as names". Indeed, quantifying nominal forms is easier, e.g., "8% of Landings" is more intuitive than "8% of To_land" or "8% of Land". Although in English most processes also have a nominal form as name, a "lexical normalization" BP is to *use the gerund form of a verb as name (as in "a Singing" or "a Representing") and leave nominal forms to non-process types.* This helps understanding the KRs and hence may avoid incorrect KR additions or re-uses (not all can be detected automatically) and therefore also *undetected redundancies*. The above cited non-process types often are about the object, agent or result of a process, e.g., Song, Definer, Representation.
- This prevents the creation of "concept types with adjectives as names", e.g., Abstract, Relative. Like types with verbs as names, the meaning of their quantification is not intuitive. Furthermore, such names do not permit to know if they refer to a particular "attribute or measure" or to a thing that has such an attribute or measure. The last case is the most frequent: the author forgot to add "_thing" at the end, as in Relative_thing.

Finally, it is also a BP to avoid using plural nominal expressions for concept/relation types, e.g., "Parents" or "r__parents". Indeed, they imply using collection types and the meaning of quantifying them is not intuitive. Using quantifiers on singular forms lead to KRs that are more precise and comparable.

6. Relating to other ODPs

To be adopted, knowledge sharing ODPs should be well inter-related by semantic relations to help people know about them and the criteria or advantages they fulfill, and thus select the ones they want to look for or commit to. Then, tools can check or enforce these commitments, or then retrieve ontologies satisfying them.

Thus, ideally, ODPs should at least be organized into categories related by specializations and exclusion relations, as in the ontology presented in Table 3. However, this is not easy. The most organized of current ODPC or BP repositories [15] seems to be ODPC. It organizes its ODPs into a specialization hierarchy with a first level of six categories. Each of them has 0 to 3 sub-levels. These six categories and their current content are:

- Content ODP: 101 ontologies, some having only a few types.
- Reasoning ODP: no ODP has yet been submitted in this category about making inferences.
- Structural ODP: 1 in the "architectural ODP" category (BPs about the structure of an ontology, e.g., the use of subtype partitions, i.e., unions of disjoint types as in Table 3) and 13 in the "logical ODP" category (translations between constructs from KRLs of different expressiveness).
- Correspondence ODP: 12 in the "Reengineering ODP" category (meta-model transformation rules to create ontologies from structured but less formal and semantic sources) and 13 in the "Alignment ODP" category (examples of relations between two elements from different ontologies).
- Lexico-Syntactic ODP: 20 linguistic structures for extracting KRs or displaying them (as with a controlled language).
- Presentation ODP: no submission of ODP has yet been submitted in this category about the usability and readability of ontologies. It has two subcategories: "Annotation ODP" and "Naming ODP".

These categories are not exclusive. An ODP can be placed in several of them. For instance, the ODPs listed in the sections 2, 3 and 4 seem to be architectural ODPs as well as logical ODPs and, for some of them, also Content ODP (like DOLCE+DnS-Ultralite is). The ODPs in Section 5 are Naming ODPs but are also related to structural ODPs.

Since there are multiple categorization possibilities, different persons will search or add a same ODP in different categories, thus leading to less relations between the ODPs and more *undetected redundancies* (as noted in the previous sections). This structure also does not lead ODP providers to collaboratively build a finely organized hierarchy or graph of ODPs. Such a structure could be obtained by formally representing each ODP as a process, using a same base ontology, e.g., the MSO (hence with the types shown in Table 3 and Table 4 as top-level types). Most of the subtype relations between ODPs could then be automatically calculated. Although this approach would scale well, such a formal and homogenous representation would be a huge work and would require quite motivated ODP providers.

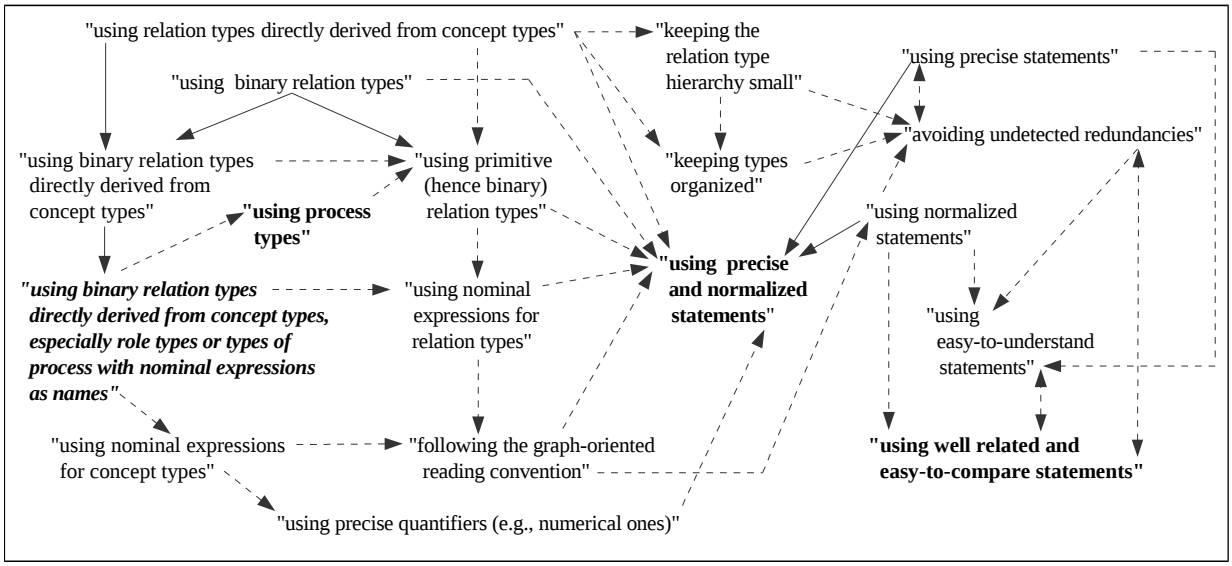
Furthermore, relations to criteria and advantages would still probably not be sufficient since relating ODPs to criteria – or process representing these criteria – is difficult. Therefore, for the ODPs advocated in this article, another approach has been adopted: i) manually setting subtype relations between ODPs (represented as process types) when this was possible, and ii) using positive "gradual pattern" relations. Table 5 is the result.

These last relations represent rules of the form "the more X, the more Y" ([1] gives a formalization). Arrows with dashed lines are positive "gradual pattern" relations. E.g., the dashed arrow from "keeping the types organized" to "avoiding undetected redundancies" can be read "the more 'keeping the types organized' is achieved, the more 'avoiding undetected redundancies' is achieved". A simpler but less direct way to read this is: "the more 'the types are kept organized', the more 'undetected redundancies are avoided' ".

This last particular rule refers to the idea that was mentioned again two paragraphs ago and which could be rephrased as: "the more a KR (type or statement) has a 'unique place' [6] in a hierarchy of KRs, the less chances there are that another person will add an equivalent KR in another place". For example, as opposed to subtype hierarchies, taxonomies relate objects (terms, documents, ...) with relations which are neither typed nor formal. Thus, people use these relations for representing subtypes, parts, instances, agents, etc. This leads to hierarchies that are difficult to search and that often have redundancies. When subtype partitions are used, this is far less the case. This is also far less the case when the hierarchy is automatically built based on the definition of each type. Like subtype relations, gradual pattern relations are typed and transitive. Hence, if used correctly, each KR in them can have a "unique place" [6], even when such relations do not form a partial order. Indeed, there is no partial order when some relations are bidi-

Table 5

Relation between the ODP advocated here (the process that has a name in italic bold characters) and related ODPs
(notes: arrows with continuous lines are subtype relations, arrows with dashed lines are positive "gradual pattern" relations;
arrows inherited via subtypes relations are left implicit, e.g., those inherited by "using precise and normalized statements")



rectional (there are some bidirectional relations in Table 5). However, gradual pattern relations permit less automatic checking possibilities than subtype partitions.

Given the explanations provided in the previous sections, the relations in Table 5 should now be understandable. *Note to the reviewers: if more details are needed, please tell us which relations from Table 5 should be explained.*

The use of gradual pattern relations between ODPs or BPs is original. The direct setting of subtype relations between them also seems original.

7. Conclusion

Knowledge sharing is difficult. It implies satisfying many criteria – and following various BPs – which, as Table 5 showed, are inter-related. To provide such BPs and ways to follow them, this article focused on the idea of deriving relation types from concept types and showed its relationships to various BPs and ODPs for knowledge modeling and sharing. Some of these BPs and ODPs were already known, several were original. In this domain, most BPs can actually be seen as ODPs. E.g., [15] lists the W3C repository of BPs (guidelines, ...) as an ODP repository.

This article also provided various *kinds* of ODPs. According to the categories of ODPC, these are architectural, logical, content and naming ODPs. However, given their inter-relations and the focus on derivation mechanisms, it is also true that this article focused on one ODP.

The proposed BPs and ODPs are applied to – and supported by – the MSO (more than 75,000 categories) which is accessible and updatable via the WebKB shared knowledge base server. Together, they help people and automated agents create KRs that are more normalized, inter-related, comparable and understandable. Furthermore, the multi-source nature of the MSO would help applying the proposed content ODPs to other ones such as DOLCE+DnS-Ultralite.

Finally, the following of the proposed BPs can easily be tested, interactively (as within WebKB) or via SPARQL queries on an ontology (e.g., it is easy to test if each relation type is defined with respect to one concept type). This makes these BPs usable as criteria for selecting ontologies.

This work will be extended by relating knowledge sharing techniques, BPs and criteria (including security related criteria, although represented as processes), via specialization relations and gradual pattern relations, positive ones as well as negative ones ("the more X, the less Y"). The focus will be on repre-

senting various approaches to knowledge sharing, e.g., those based on formal documents, those based on collaborative editing within a shared ontology server and those based on knowledge exchange between ontology servers. Thanks to the specialization relations and the positive/negative gradual pattern relations, the various kinds of ways to share knowledge and their respective advantages and drawbacks should be clearer.

References

[1] S. Ayouni, A. Laurent, S. Ben Yahia and P. Poncelet, Mining closed gradual patterns, in: proceedings of ICAISC 2010, 10th international conference on Artificial intelligence and soft computing, pp. 267-274, Springer-Verlag Berlin, Heidelberg.

[2] J. Barwise, J. Gawron, G. Plotkin and S. Tutiya, Situation Theory and its Applications (Vol. 2), CSLI Lecturre Notes no. 26, CSLI publications, Stanford, CA, 1991, 622 pages.

[3] S. Borgo and C. Masolo, Ontological Foundations of DOLCE, in the Handbook on Ontologies (Second Edition), S. Staab and R. Studer, ed., Springer Verlag 2009, pp. 361-382.

[4] A.Carnie, Syntax: A Generative introduction. 2nd Edition. Blackwell Publishers, 2006.

[5] J. Correia and R. Pöschel, The Teridentity and Peircean Algebraic Logic, in: LNCS 4068/2006, Springer Berlin, pp. 229-246, http://www.springerlink.com/content/42v7r03v4x08l831/

[6] G. Dromey, Scaleable Formalization of Imperfect Knowledge, in: proceedings of AWCVS 2006, Macau, China.

[7] A. Gangemi, DOLCE+DnS-Ultralite, RDF+OWL ontology at http://www.ontologydesignpatterns.org/ont/dul/DUL.owl

[8] A. Gangemi, N. Guarino and A. Oltramari A., Restructuring Wordnet's Top-Level. AI Magazine, 40(5), 2002, pp. 235-244.

[9] M. Genesereth and R. Fikes, Knowledge Interchange Format, Version 3.0, Reference Manual. Technical Report Logic-92-1, Computer Science Dept., Stanford University. http://www.cs.umbc.edu/kse/

[10] N. Guarino, C. Masolo and G. Vetere, Ontoseek: Content-based Access to the Web, IEEE Intelligent Systems, Vol. 14, No. 3, 1999, pp. 70-80.

[11] Ph. Martin, Collaborative knowledge sharing and editing, IJCSIS, Volume 6, Issue 1 (2011), pp. 14-29.

[12] Ph. Martin, Correction and Extension of WordNet 1.7, in: LNAI 2746, pp. 160-173. See also http://www.webkb.org/doc/MSO.html

[13] R. Mizoguchi, K. Kozaki and Y. Kitamura, Ontological Analyses of Roles, in: proceedings of the IEEE FedCSIS 2012, pp. 489–496.

[14] I. Niles and A. Pease, Towards a standard upper ontology, in: proceeding of FOIS 2001, ACM, pp. 2-9.

[15] M. Poveda-Villalón, M.C. Suárez-Figueroa and A. Gómez-Pérez, Reusing Ontology Design Patterns in a Context Ontology Network, in: proceedings of WOP 2010, CEUR-WS.org volume 671, pp. 35-49.

[16] V. Presutti and A.Gangemi, Content Ontology Design Patterns as Practical Building Blocks for Web Ontologies, in: proceedings of ER 2008, Spaccapietra S. et al., ed.

[17] RIF-FLD, RIF Framework for Logic Dialects (2nd Edition), W3C Recommendation of 2013, H. Boley and M. Kifer, ed., http://www.w3.org/TR/2013/REC-rif-fld-20130205/

[18] J.F. Sowa, Conceptual Graphs Summary, in: "Conceptual Structures: current research and practice", Ellis Horwood, 1992, pp. 3-51.

[19] J.F. Sowa, Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., 2000, 594 pages. See also http://www.jfsowa.com/ontology/toplevel.htm

[20] C. Welty, Context Slices, 2010, http://ontologydesignpatterns.org/wiki/Submissions:Context_Slices

[21] G. Zarri, Representation and Management of Narrative Information: Theoretical Principles and Implementation, Springer 2009, Series: Advanced Information and Knowledge Processing, 312 pages.