Categorizing or Generating Relation Types and Organizing Ontology Design Patterns

Philippe Martin, Jérémy Bénard

Categorizing or Generating Relation Types and Organizing Ontology Design Patterns

Philippe A. Martin
University of La Réunion, EA2525 LIM,
Saint-Denis de la Réunion, F-97490 France
+ adjunct researcher of the School of I.C.T. at
Griffith University, Australia
Email: Philippe.Martin@univ-reunion.fr

Jérémy Bénard
Uni. of La Réunion, EA2525 LIM and GTH, Logicells,
55 rue Labourdonnais, 97400 Saint-Denis, France
Email: Jeremy.Benard@logicells.com

***Abstract*—This article proposes an ontology design pattern leading knowledge providers to represent knowledge in more normalized, precise and inter-related ways, hence in ways that help automatic 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. This pattern is: "using binary relation types directly derived from concept types, especially role types or process types". The article explains this pattern and relates it to other ones, thereby illustrating ways to organize such patterns. It also provides a top-level ontology for generating relation types from concept types, e.g., those from lexical ontologies such as those derived from the WordNet lexical database. This generation and categorization helps normalizing knowledge, reduces having to introduce new relation types and helps keeping all the types organized.**

I. Introduction

ONTOLOGY Design Patterns (ODPs) are "modeling solutions to solve a recurrent ontology design problem" [1]. A "Conceptual ODP" describes a best practice (BP) for content modelling [1]. Since we only consider ODPs that represent BPs, we use these two terms interchangeably in this article to ease its reading. Many ODPs have been described. E.g., about 160 are registered in the ODP catalog at http://ontologydesignpatterns.org which, in this article, will now be referred to as ODPC. Despite these ODPs, most of thousands of existing ontologies that exist are still poorly inter-connected and heterogeneous in their design. It is then difficult for people and 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 modeling and sharing – as opposed to knowledge exploitation with computational tractability constraints – and, more precisely, specifically aimed 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. 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 relates it to other ones, via specialization relations and *gradual pattern* relations. This 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 process types"*. 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. The sections 2, 3 and 4 explain, formalize and illustrate the different parts of ABP. Section 5 relates them to other ODPs and thereby also gives more rationale.

II. Using Binary Relations

ABP starts by advocating the use of *binary relations*, i.e., logical statements based on binary predicates. In the RDF model, these statements are called *triples* and binary relation types are called *properties*. In this article, types that are not relation types (RTs) are refered to as *concept types* (CTs), i.e., *classes* in the RDF model. The expression *concept individual* will be used for anything that is neither a type nor a relation.

Since ABP is language independent, this article uses a general terminology, one compatible with those for Conceptual Graphs and RIF-FLD [2], the W3C Framework for Logic Dialects of the Rule Interchange Format. 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. For clarity purpose too, in the examples, RT 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.

This work was not supported by any organization.

Following ABP does not prevent using non-binary RTs as long as definitions or rules are also provided to enable the automatic translation of "KRs using non-binary RTs" into "KRs using binary RTs". Table I illustrates such rules for various kinds of use cases but only the third row is also about the focus of Section 3, i.e., deriving a RT from a CT.

One reason why such definitions or rules are useful for knowledge sharing is that binary relations can be *compared* while two relations of different arities generally cannot. *Two types or KRs are comparable if and only if an equivalence or specialization relation between them has been directly stated or can be automatically 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 I, some relations of different arities can be translated into binary relations using a list as destination. Then, they can be *compared*.

A *related reason* why such definitions or rules are useful for knowledge sharing is that they make more information explicit. As detailed it Section 5, normalizing knowledge, enhancing its comparability or adding more information have strong relationships.

In practice, with a KR language (KRL) allowing *contexts* and sets or lists, it is easy to avoid the use of relations with

TABLE I.
EXAMPLES OF HOW TO DEFINE A GIVEN RT WITH RESPECT TO OTHER TYPES
(THE RIF-FLD PS NOTATION IS USED IN THE NON-HIGHLIGHTED PARTS; VARIABLES BEGIN BY "?"; ":-" MEANS "<=")

If you wish to (re-)use non-binary RTs, 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 RTs 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 RTs representing types of processes, as in

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

instead of using classic primitive binary RTs 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) ) )
```

arity greater than two. A *context*, i.e., a *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 is not restricted to first-order logic, it lacks a construct for expressing contextualizations in simple ways, as in KIF [3] for example. However, the second row of Table I 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 [4]. It relies on introducing *concept individuals within contexts* 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 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, *for the purpose of knowledge modeling and sharing* – as opposed to knowledge exploitation which comes after and may require converting the knowledge into KRLs of reduced expressiveness but which can be handled efficiently – *a BP is to i) use a KRL that handles contexts or use more 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 that deal 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.

To conclude, although formally specifying the semantics of *relations of arity greater than two* requires at least one primitive ternary relation [5], in practice there is no necessity to use such relations for knowledge modelling.

There is no claim here that the idea of "translating non-binary RTs 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 necessity of using constructs supporting different kinds of contexts [6], and iii) this best practice is sometimes unknown to users of KRLs allowing non-binary relations.

III. Deriving Relation Types from Concept Types

ABP advocates the use of – or specifications of translations into – binary RTs *directly derived from CT*". A CT may have multiple *directly derived RTs* if they have *un-comparable* signatures, i.e., if none specializes another one. The third row of Table I illustrated a way to directly derive an RT from a CT using a rule and a relation of type r__directly_derived_relation. The first two rows illustrate the definitions of non-binary RTs mainly with respect to binary RTs. This is useful as an intermediary step: the final step – deriving these last binary RTs from a CT, e.g., one named *List_of_surrounding_entities* – was not illustrated in Table I.

Manually or automatically defining each RT with respect to a CT makes additional 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 prevents some knowledge providers to develop distinctions only in the RT hierarchy while others develop distinctions only in the CT hierarchy, thus leading to *undetected redundancies* within a shared knowledge base or in different ontologies. Second, it ensures that any distinction can be used – *without losing possibilities of knowledge representation and matching* – 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) CTs are easier to organize 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. Thus, both cases lead to better categorizations in the concept and relation hiearchies.

These advantages of using *defined RTs* come for free when 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 at all. This last option was used in the knowledge server Ontoseek [7] and is used in the knowledge base server WebKB (www.webkb.org; [8]). 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. 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*. Table II illustrates the approach and then gives rules that would actually generate the derived RTs. The next section complements this framework by giving an ontology of the CTs these rules can be applied to. These RT generation rules permit to formalize the framework. They rely on the functions *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 [3]. In WebKB, such rules are not actually executed but a more efficient process relying on the same idea is used. Indeed, during the parsing of statements, whenever a CT is used where a RT is 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 in the framework described by Table II, 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.

TABLE II.
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
(NOTE: IN THESE EXAMPLES, THE TYPES CREATED BY THE AUTHORS OF THIS ARTICLE HAVE NO PREFIX TO INDICATE THEIR NAMESPACE).

Table I gave examples of how a rule can define a RT with respect to a CT. This had to be done for each RT. Here, the approach is simpler. 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 the definitions given in the next row of this table, the derived RT is automatically created.
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))
//-> associates a signature to the CT Father and derives the RT r__father with 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_that_thing_as_destination
// "##" means "is subtype of"; "#" means "is instance of"; this rule derives the expected RT for each
// subtype of Thing_usable_for_deriving_a_binary_relation_with_that_thing_as_destination

Forall ?processType Exists ?r
And (?r = **f__ derived_binary_relation** (?processType List (Agent Object))
Forall ?process ?agent ?object And (r__agent (?process ?agent) r__ object (?process ?object))
:- And (?process # ?processType ?r (?agent ?object)))
:- ?processType ## Process *//this rule 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 //"##" means "is subtype of"
          :- 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 ) ) )
  :- r__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"; owl:FunctionalProperty is a 2nd-order type
  :- r__signature_for_derived_functional_binary_relation ( ?t List ( ?t_domain ?t_range ) )
```

IV. DERIVING FROM ROLE TYPES OR PROCESS TYPES

ABP advocates the derivation of RTs from CTs, "*especially role types or process types*". The third row of Table I illustrated this for processes. In this article, a *process* refers to a *situation* that is not a *state*, and hence that makes a change. A *situation* is something that *occurs* in a real/imaginary region of time and space. These conceptual distinctions come from the *Situation Semantics* [9] and are the basis of John Sowa's first top-level ontology [10]. There are re-used in this article for at least the following reasons:

- They are rather intuitive and generalize other well known types. E.g., *Perdurant* from Dolce [11] is subtype of Process.

- They are very adequate for the signatures of thematic relations [12], e.g., r__agent, r__recipient, r__cause, r__instrument. Such types are top-level types of relations from a process.
- In this article, a *role type* (e.g., Agent, Experiencer, Recipient, Cause, Instrument) is a CT which is defined – or could be defined – as being the range of a thematic RT. This informal definition of a role is a bit more general than what is usually thought to be a role type [13] but here it is sufficient: as defined in this article, *processes* and *role* types can be used for deriving CTs into binary RTs.
- Thematic RTs or their subtypes can also be used for defining most RTs. Thus, doing so normalizes KRs.
- Most statements implicitly or explicitly refer to a process. Representing it, either directly or via RTs directly derived from a process, strongly normalizes KRs. Not doing so, which unfortunately is the case in many ontologies, amounts to losing precisions and many KR comparison possibilities.

Fig. 1 compares *CTs usable for directly deriving a binary RT* with other types. The common supertype of these CTs is Thing_usable_for_directly_deriving_a_binary_relation. Only its subtypes can be used for deriving binary RTs; this

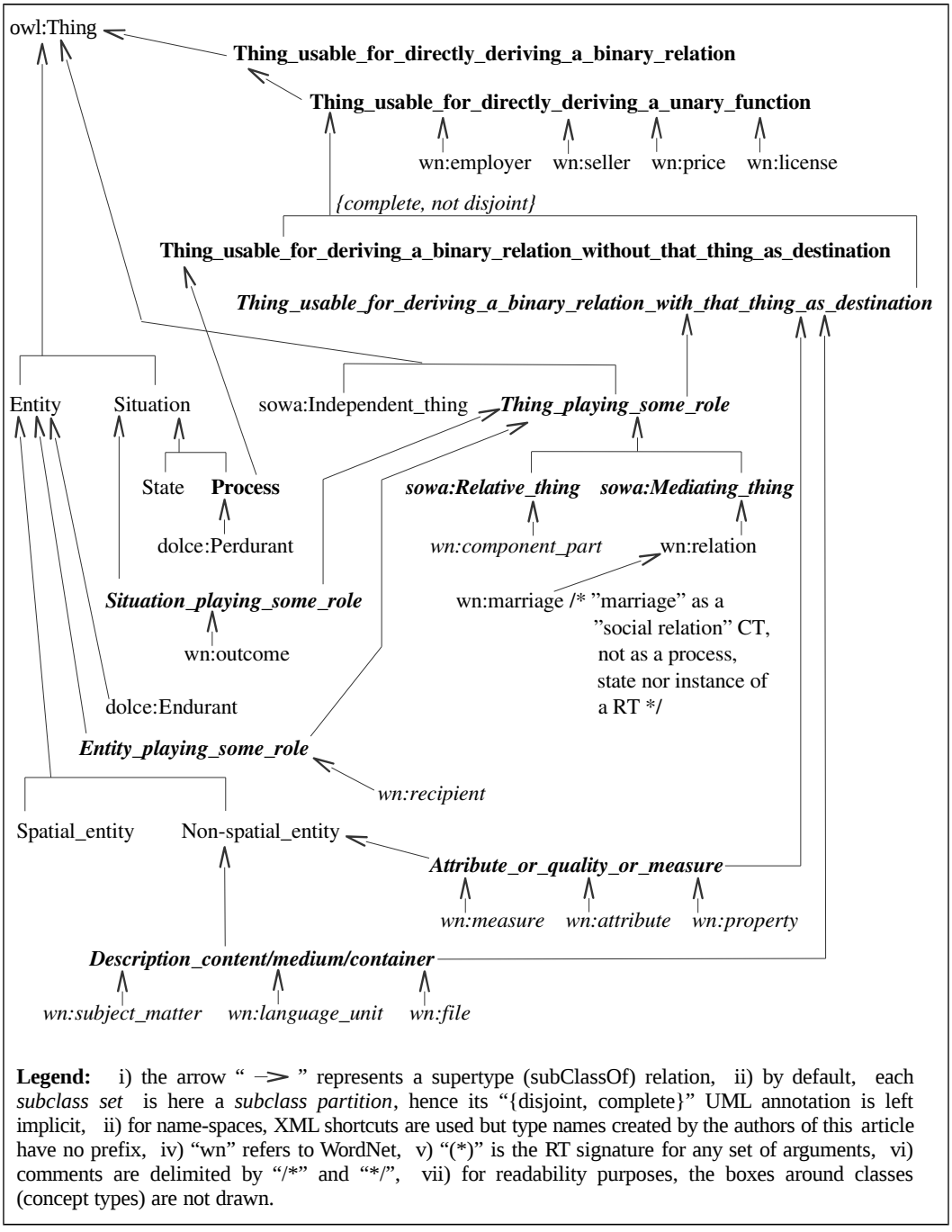


Fig. 1. *Slightly adapted UML* representation of a subtype hierarchy to compare the type Thing_usable_for_directly_deriving_a_binary_relation with other types.

includes types for processes and roles. Fig. 2 illustrates subtype relations between such derived RTs. Fig. 3 displays common top-level types for relations from a process, most of which are thematic RTs. Fig. 3 re-uses top-level types shown in Fig. 1. All the types in these figures are part of the *Multi-Source Ontology* (MSO [14]) which is accessible and cooperatively updatable via WebKB. Hence, the names in these figures are names accessible via this Web server. However, these figures have not previously been published.

The MSO includes more than 75,000 categories and relates them by more than 100,000 relations, mainly subtype relations. It categorizes WordNet types as well as types from various top-level ontologies (DOLCE included) with respect to the types shown in Fig. 1 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 Fig. 1 or specializations of them. Thus, in the subtype hierarchy of the MSO for *things usable for directly deriving a binary relation*, there are currently more than 4800 process types, 2900 role types (for *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 relations without having to declare new RTs. The 4800 process types can also be used directly with *relations from a process*. Finally, the types shown in Fig. 3 for these relations can implicitly or explicitly be specialized by 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) the MSO can be complemented online via WebKB.

In Fig. 1, the types named Relative_thing and Mediating_thing come from John Sowa's second top-level ontology [15].

To show how rules can be used to associate a signature to a CT and thereby to a derived RT, examples in Table II 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*". Fig. 2 shows how the various relations types – derived or not from CTs – 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 Fig. 2, 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 for definitions, hence for signatures.

ODPC includes the DOLCE+DnS-Ultralite ontology [16] and categorizes it as *Content ODP*. ODPC also includes related but smaller *content ODPs* such those named *ActingFor* and *Agent-Role*. Its *DnS (Descriptions and Situations)* part includes some types which can be seen as subtypes of those in Fig. 3. ODPC proposes many RTs which could be – but, it seems, are not – derived from process types, e.g., RTs with names such as *actsFor*, *conceptualizes* or *defines*. Yet, some of its CTs have been aligned with OntoWordNet [17]. Thus, the ontology and approach proposed in this section and the previous one could be used to extend and normalize DOLCE+DnS-Ultralite. This would support more KR comparison possibilities.

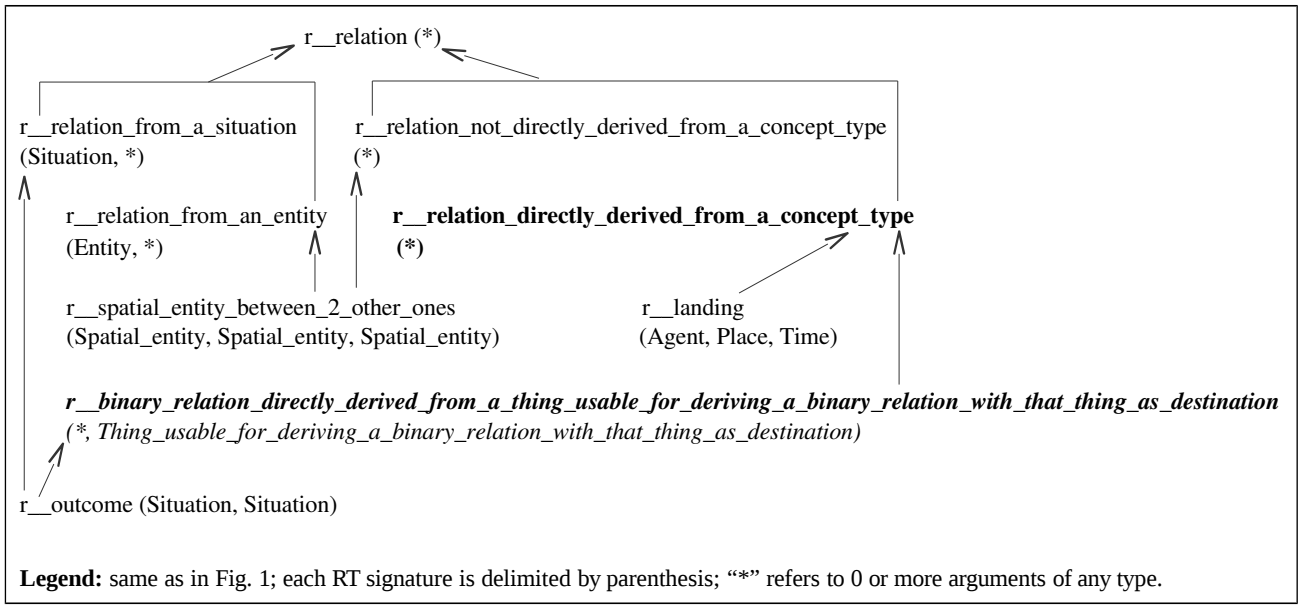


Fig. 2. Subtype hierarchy of some relation types derived from subtypes of the concept type Thing_usable_for_directly_deriving_a_binary_relation.

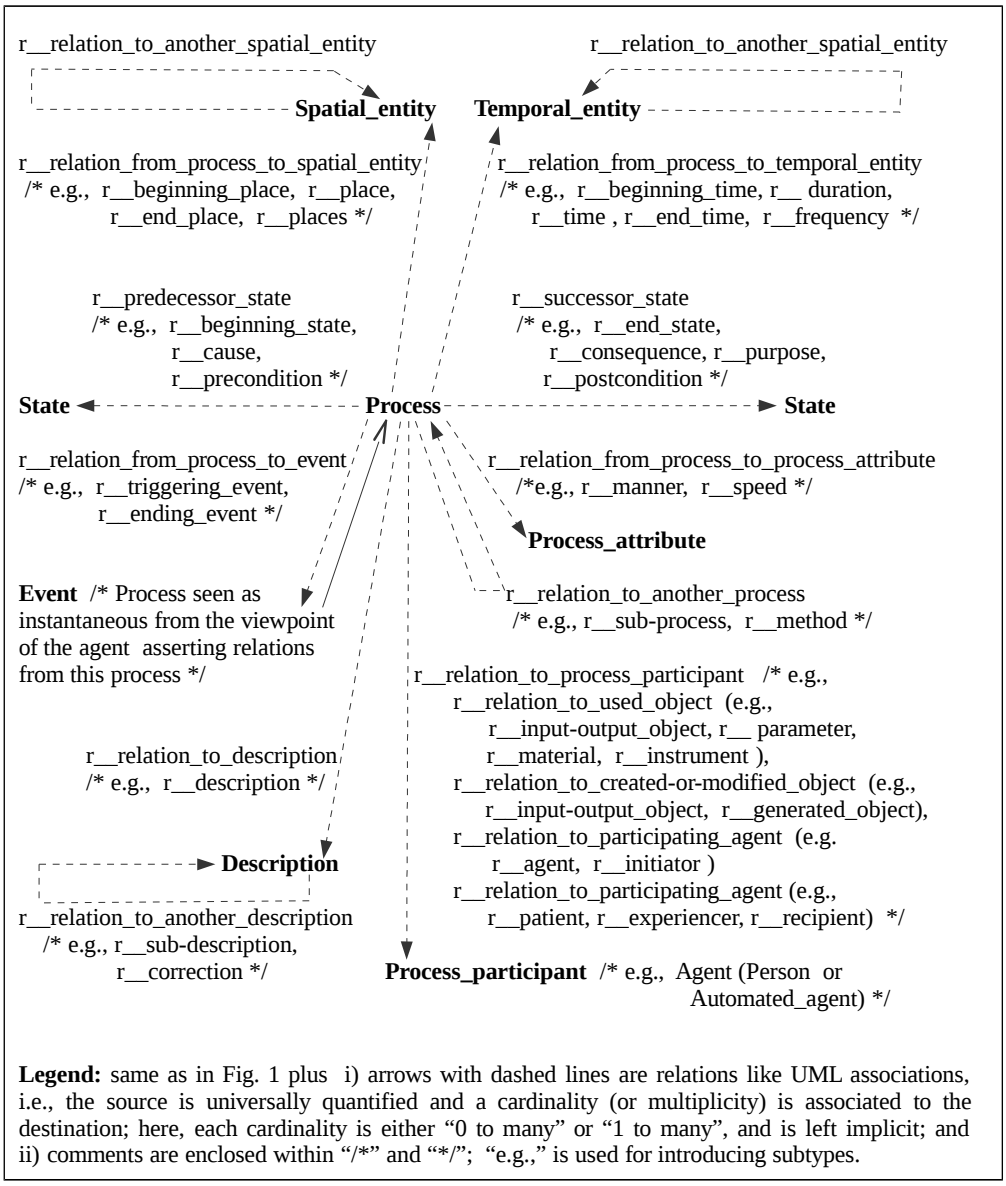


Fig. 3. Examples of common types of relations from a process; most of them are thematic RTs.

V. 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. Thus, people can search and select ODPs to commit to. Then, tools can check or enforce these commitments, or retrieve ontologies satisfying them.

Thus, ideally, ODPs should at least be organized into categories related by specializations and exclusion relations, as in the hierarchy presented in Fig. 1. However, this is not easy. The most organized of current ODPC or BP repositories [18] 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.
- *Structural ODP*: 1 ODP 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 Fig. 1 – 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 – these ODPs are examples of RTs between 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*.

All these categories are not exclusive. An ODP can be placed in several of them. E.g., 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, e.g., the DOLCE+DnS-Ultralite. The ODPs we gave in Section 5 are Naming ODPs but are also related to Structural ODPs.

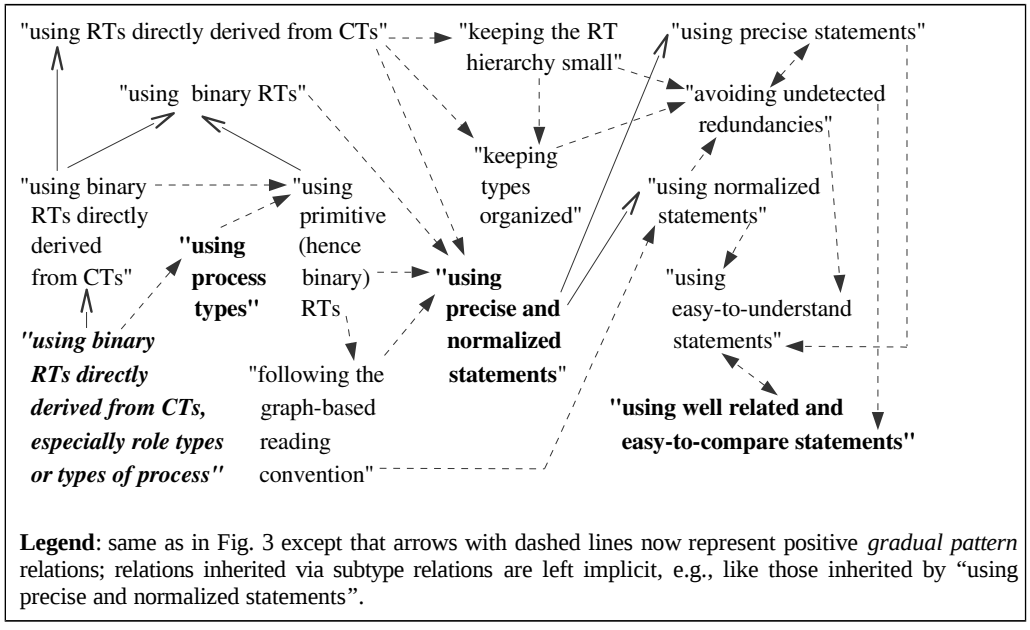
Since there are multiple categorization possibilities, different persons will search or add a same ODP in different categories, thus leading to less relations between the ODPs and more *undetected redundancies*, as noted in the previous sections. This structure also does not lead ODP providers to collaboratively 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 Fig. 1 and Fig. 3 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 processes 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 or BPs represented as process types, and ii) using positive *gradual pattern* relations. Fig. 4 is the result.

These last relations represent rules of the form “the more X, the more Y”. [19] gives a formalization for such relations. 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´ ”.

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´ [20] in a hierarchy of KRs, the less chances there are that another person will add an equivalent KR in another place”. E.g., 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 can have a *unique place* [20] in the graph formed by these transitive relations. However, gradual pattern relations do not enable as many automatic checking possibilities as subtype partitions.

Given the explanations provided in the previous sections, the relations in Fig. 4 should now be understandable. The use of gradual pattern relations between ODPs or BPs is original. The direct setting of subtype relations between them also seems original.



Legend: same as in Fig. 3 except that arrows with dashed lines now represent positive *gradual pattern* relations; relations inherited via subtype relations are left implicit, e.g., like those inherited by “using precise and normalized statements”.

Fig. 4. Supertype relations and *gradual pattern* relations between ABP (the BP advocated in this article; see the BP name in italic bold characters at the bottom left of the figure) and related BPs.

VI. Conclusion

Knowledge sharing is difficult. It implies satisfying many criteria – and following various BPs – which, as Fig. 4 showed, are inter-related. To provide such BPs and ways to follow them, this article has focused on the idea of deriving RTs from CTs and has shown the relationships between this ODP to other ones for knowledge modeling and sharing. Some of these ODPs were already known, several were original.

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 one named ABP – composed of simpler ODPs.

The ODPs we proposed are applied to – and supported by – the MSO which includes more than 75,000 categories and which is accessible and updatable online via the WebKB shared knowledge base server. Together, these resources and tools 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 ODPs can easily be tested, e.g., via SPARQL queries on an ontology or, interactively, within WebKB. For example, it is easy to test if each RT is defined with respect to one CT. This makes these BPs usable as criteria for selecting ontologies.

This work will be extended by relating knowledge sharing techniques, BPs and criteria, via specialization relations and gradual pattern relations. *Negative* gradual pattern relations – "the more X, the *less* Y" – will also be used. The focus will be on representing 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 their organization by specialization relations and gradual pattern relations, the various kinds of ways to share knowledge and their respective advantages and drawbacks should be clearer.

References

[1] A. Gangemi, and V. Presutti, "Ontology Design Patterns," *Handbook on Ontologies*, 22 May 2009, pp. 221-243, http://doi.org/10.1007/978-3-540-92673-3_10

[2] H. Boley, and M. Kifer (eds.), *RIF Framework for Logic Dialects (2nd edition)*. W3C Recommendation 2013, http://w3.org/TR/2013/REC-rif-fld-20130205/

[3] M. Genesereth, and R. Fikes, *Knowledge Interchange Format, Version 3.0, Reference Manual*. Technical Report 1992, Logic-92-1, Stanford Uni., http://www.cs.umbc.edu/kse/

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

[5] J. Correia, and R. Pöschel, "The Teridentity and Peircean Algebraic Logic," *LNCS 4068*, Springer Berlin, 2006, pp. 229–246, http://doi.org/10.1007/11787181_17

[6] G. Zarri, *Representation and Management of Narrative Information: Theoretical Principles and Implementation.* Springer, Series: Advanced Information and Knowledge Processing, 312 pages, 2009, http://doi.org/10.1007/978-1-84800-078-0

[7] 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, http://doi.org/10.1109/5254.769887

[8] P. Martin, "Collaborative knowledge sharing and editing," *IJCSIS*, vol. 6, Issue 1, 2011, pp. 14–29, http://www.worldcat.org/issn/1646-3692

[9] J. Barwise, J. Gawron, and G. Plotkin, *Situation Theory and its Applications*. CSLI publications, 2000, 655 pages, http://www.worldcat.org/oclc/947127096

[10] J.F. Sowa, "Conceptual Graphs Summary," *Conceptual Structures: current research and practice*, Ellis Horwood, 1992, pp. 3–51, http://www.worldcat.org/oclc/856836888

[11] S. Borgo, and C. Masolo, "Ontological Foundations of DOLCE," *Theory and Applications of Ontology: Computer Applications*, R. Poli, M. Healy, and A. Kameas (eds.), Springer, 2010, pp. 279–295, http://doi.org/10.1007/978-90-481-8847-5_13

[12] A. Carnie, *Syntax: A Generative introduction.* Wiley-Blackwell publishers, 2013, http://www.worldcat.org/oclc/779740455

[13] R. Mizoguchi, K. Kozaki, K., and Y. Kitamura, "Ontological Analyses of Roles," *IEEE FedCSIS 2012*, pp. 489–496, oclc: 5873174590.

[14] Ph. Martin, "Correction and Extension of WordNet 1.7," *LNAI 2746*, pp. 160–173, ICCS 2003, http://doi.org/10.1007/b11835, see also http://www.webkb.org/doc/MSO.html

[15] J.F. Sowa, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Boston : Course Technology, 2012, 594 pages, http://www.worldcat.org/oclc/819364955, see also http://www.jfsowa.com/ontology/toplevel.htm

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

[17] A. Gangemi, N. Guarino, and A. Oltramari, "Restructuring Wordnet's Top-Level," *AI Magazine*, vol. 40, no. 5, 2002, pp. 235–244.

[18] M. Poveda-Villalón, M.C. Suárez-Figueroa, and A. Gómez-Pérez, "Reusing Ontology Design Patterns in a Context Ontology Network," in Proc. *WOP 2010*, CEUR-WS.org vol. 671, pp. 35–49, http://www.worldcat.org/oclc/5495106523

[19] S. Ayouni, A. Laurent, S. Ben Yahia, and P. Poncelet, "Mining closed gradual patterns," *LNCS 6113*, ICAISC 2010, pp. 267–274, Springer-Verlag Berlin, Heidelberg, http://www.worldcat.org/oclc/3719235

[20] G. Dromey, "Scaleable Formalization of Imperfect Knowledge," in Proc. *AWCVS 2006*, Macau, China, http://www.worldcat.org/oclc/669648707