

Directly Deriving Binary Relation Types From Concept Types, especially Process or Role Types

Philippe A. Martin¹ and Jérémy Bénard²

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 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 knowledge from different sources. This pattern is 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 types of process". 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/normalization/matching, 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>" which, in this article, will now be referred to as "ODPC". However, the thousands of ontologies (UML schemas included) 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 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 – which is also a best practice (BP) – 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 types of process"*. 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 and 4 explain, formalize and illustrate the different parts of ABP. Section 5 relates them to other ODPs and thereby also gives more rationale.

2 Using Binary Relations

ABP starts by advocating the use of *"binary relations"* (e.g., "properties" in RDF and OWL). In this article, types that are not relation types (RTs) are referred to as "concept types" (CTs; "classes" in RDF and OWL). The terms "individual" or "concept individual" will be used for things that are not types nor relations.

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. 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.

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 1 illustrates such rules for various kinds of use cases (only the third row is also about the focus of Section 3: deriving a RT from a CT).

One reason why such rules are useful for knowledge sharing is that binary relations can be "compared" while 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 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".

Table 1. 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)))

A *related reason* is that they make more information explicit. Normalizing, precisising and supporting knowledge comparability have strong relationships (cf. Section 5).

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" (aka "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 [3] 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 [4]. 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 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).

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 to *generate* all possible relations" [5].

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 need for constructs supporting different kinds of contexts (e.g., [6]), and 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 RTs "*directly derived from CTs*". A CT may have multiple "directly derived RTs" if they have "uncomparable" signatures (i.e., if none specializes another one). The third row of Table 1 illustrates a way to directly derive a (binary or not) 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 a intermediary step: the final step – deriving these last binary RTs from a CT (e.g., named "List_of_surrounding_entities") – is not illustrated in Table 1.

When genuine definitions or rules are manually given for each derived RT, as illustrated in the third row of Table 1, the advantages of the approach (over directly using RTs without defining them) only come from the existence of this definition. 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 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 come for free when the RTs are automatically derived from CTs. Furthermore, doing so permits a system to i) hide the automatically derived RTs in the RT 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 [7] and is used in the shared/personal 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 but 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 2 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 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, no such rules are executed: 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 in the framework described by Table 2, 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 2. 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 1 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 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
```

Here are rules that permit such derivations. 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 ) ) )
```

4 Deriving From Role Types and Processes

ABP advocates the derivation of CTs, "*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 [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 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 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: process and role types (as defined here) 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 (and then using thematic RTs or subtypes of them) or via RTs directly derived from a process, strongly normalizes KRs. Not doing so, which unfortunately is the case in most 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 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 server. However, these figures 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 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 ("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

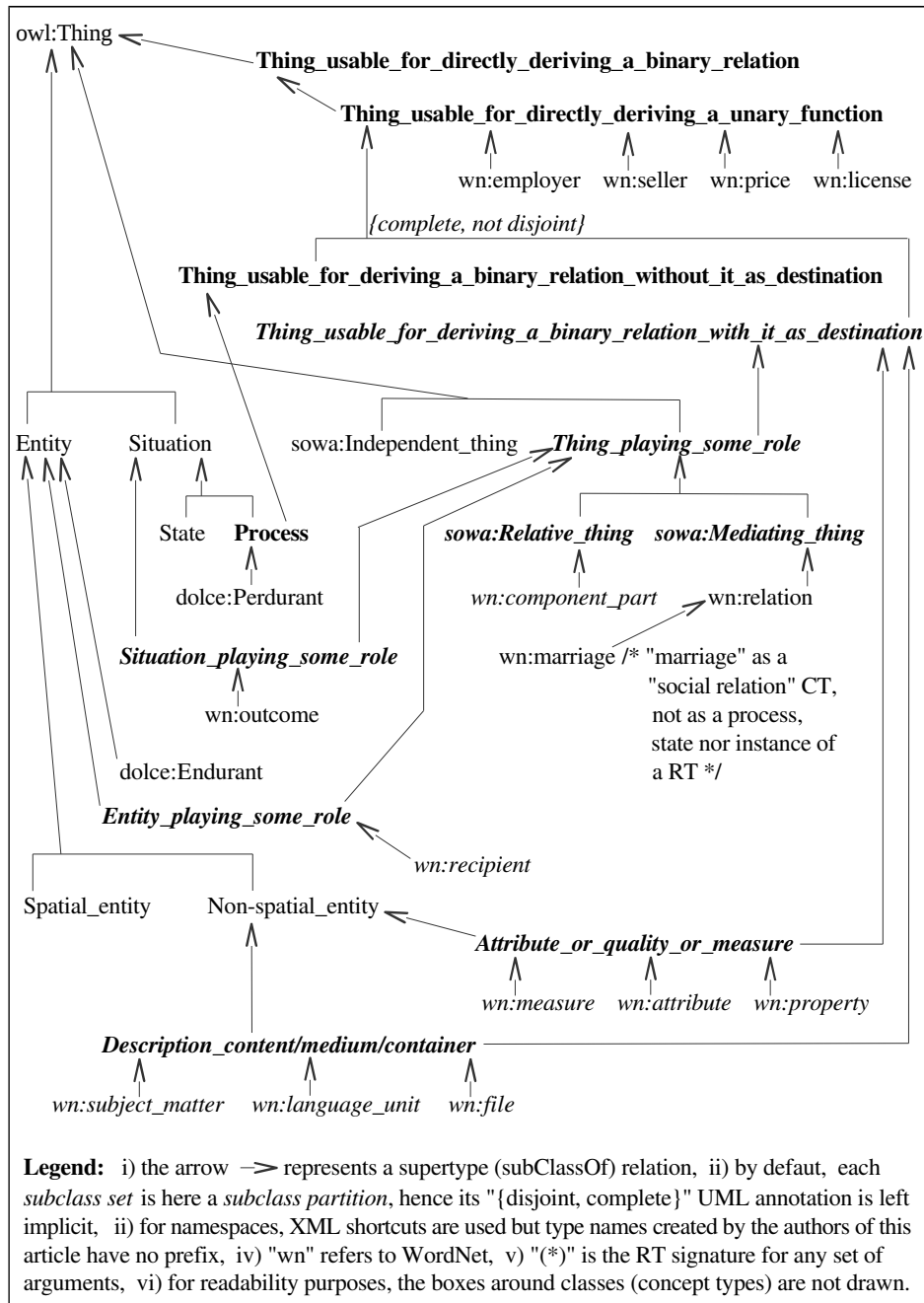


Fig. 1. Slightly adapted UML representation of a subtype hierarchy to compare the type `Thing_usable_for_directly_deriving_a_binary_relation` with other types.

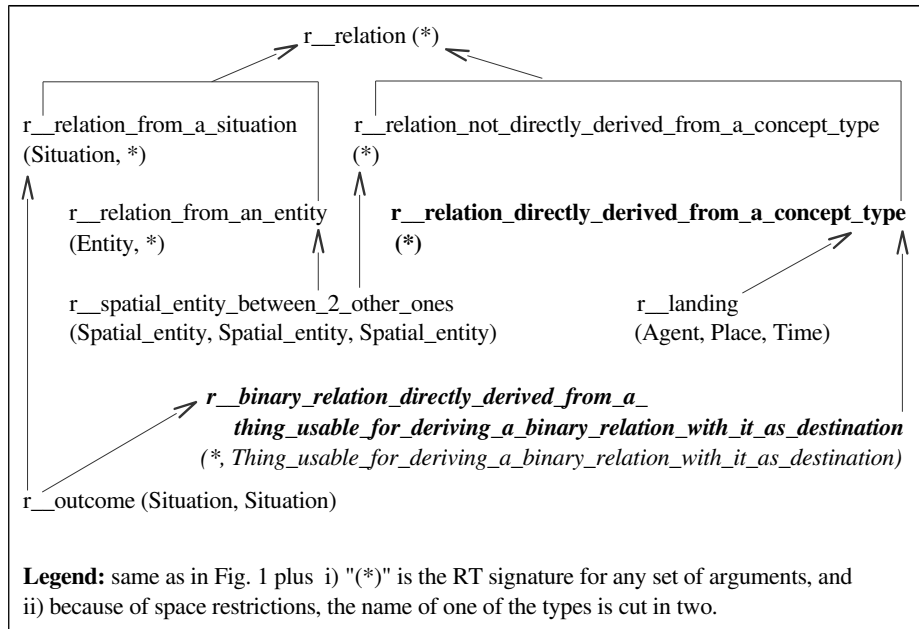


Fig. 2. Subtype hierarchy of some relation types derived from subtypes of the concept type Thing_usable_for_directly_deriving_a_binary_relation with other types.

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) 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 KRIs 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 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 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". 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 PS for definitions, hence for signatures.

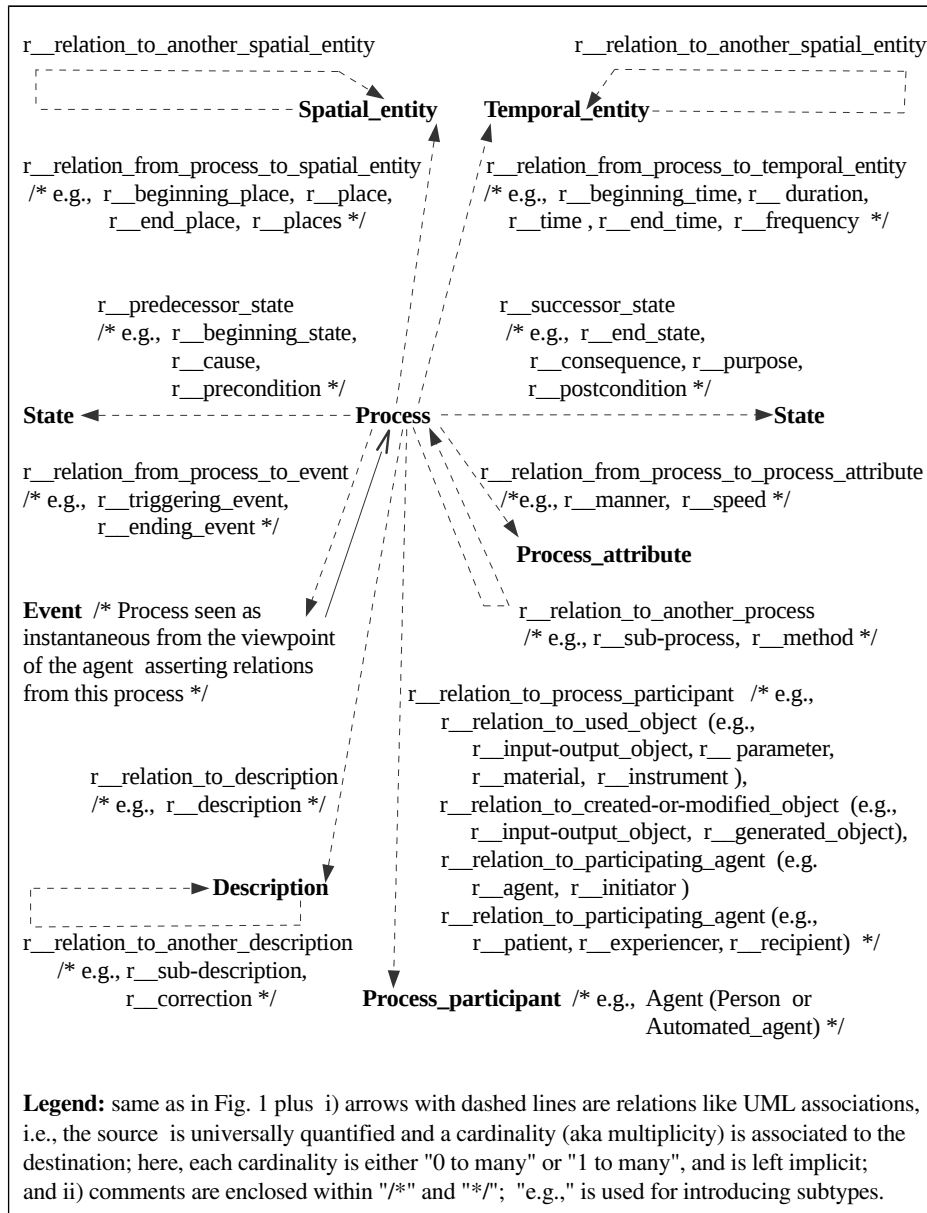


Fig. 3. Examples of common types of relations from a process; most of them are thematic RTs.

ODPC includes the DOLCE+DnS-Ultralite ontology [16] as "content ODP" as well as smaller "content ODPs" extracted from it, e.g., "ActingFor" and "Agent-Role". Its DnS (Descriptions and Situations) part includes some types which can be seen as subtypes

of those in Fig. 3. It proposes many relations types 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 DOLCE+DnS-Ultralite. This would support more KR comparison possibilities.

5 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 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 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 (they are examples of RTs 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. 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 (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 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 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. Fig. 4 is the result.

These last relations represent rules of the form "the more X, the more Y" ([19] 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".

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 in them can have a "unique place" [20], even when such relations do not form a partial order. Indeed, there is no partial order when some relations are bidirectional (there are some bidirectional relations in Fig. 4). However, gradual pattern relations permit less automatic checking possibilities than 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.

6 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 focused on the idea of deriving RTs from CTs and showed its relationships to various BPs or ODPs for knowledge modeling and sharing. Some of these BPs or ODPs were already known, several were original. In this domain, most BPs can actually be seen as ODPs. E.g., [18] 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

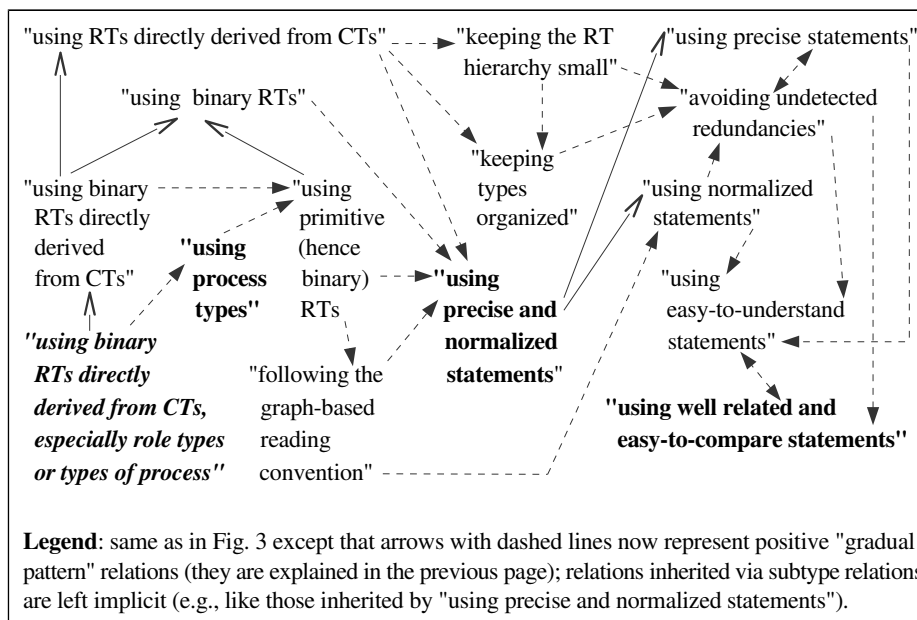


Fig. 4. Supertype relations and "gradual pattern" relations between the ODP advocated in this article (cf. the name in italic bold characters) and related ODPs.

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 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 (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 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 positive/negative gradual pattern relations, the various kinds of ways to share knowledge and their respective advantages and drawbacks should be clearer.

8 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. 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)
4. Welty, C.: Context Slices. http://ontologydesignpatterns.org/wiki/Submissions:Context_Slices (2010)
5. Correia, J., Pöschel, R.: The Teridentity and Peircean Algebraic Logic. In: LNCS 4068, Springer Berlin, pp. 229–246 (2006)
6. Zarri, G.: Representation and Management of Narrative Information: Theoretical Principles and Implementation. In: Springer, Series: Advanced Information and Knowledge Processing, 312 pages (2009)
7. Guarino, N., Masolo, C., Vetere, G.: Ontoseek: Content-based Access to the Web. IEEE Intelligent Systems, vol. 14, No. 3, pp. 70–80 (1999)
8. Martin, Ph.: Collaborative knowledge sharing and editing, IJCSIS, vol. 6, Issue 1, pp. 14–29 (2011)
9. Barwise, J., Gawron, J., Plotkin, G., Tutiya, S.: Situation Theory and its Applications (Vol. 2). CSLI Lecture Notes no. 26, CSLI publications, Stanford, CA, 622 pages (1991)
10. Sowa, J.F.: Conceptual Graphs Summary. In: "Conceptual Structures: current research and practice", Ellis Horwood, pp. 3–51 (1992)
11. Borgo, S., Masolo, C.: Ontological Foundations of DOLCE. Handbook on Ontologies (Second Edition), S. Staab and R. Studer (eds.), Springer Verlag, pp. 361–382 (2009)
12. Carnie, A.: Syntax: A Generative introduction. 2nd Edition. Blackwell Publishers (2006)
13. Mizoguchi, R., Kozaki, K., Kitamura, Y.: Ontological Analyses of Roles. In: IEEE FedCSIS 2012, pp. 489–496.
14. Martin, Ph.: Correction and Extension of WordNet 1.7. In: LNAI 2746, pp. 160–173. See also <http://www.webkb.org/doc/MSO.html>
15. Sowa, J.F.: Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., 2000, 594 pages. 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. Gangemi, A., Guarino, N., Oltramari A.: Restructuring Wordnet's Top-Level. AI Magazine, 40(5), pp. 235–244 (2002)
18. Poveda-Villalón, M., Suárez-Figueroa, M.C., Gómez-Pérez, A.: Reusing Ontology Design Patterns in a Context Ontology Network. In: WOP 2010, CEUR-WS.org vol. 671, pp. 35–49.
19. Ayouni, S., Laurent, A., Ben Yahia, S., Poncelet, P.: Mining closed gradual patterns. In: ICAISC 2010, 10th international conference on Artificial intelligence and soft computing, pp. 267–274, Springer-Verlag Berlin, Heidelberg.
20. Dromey, G.: Scaleable Formalization of Imperfect Knowledge. In: AWCVS 2006, Macau, China.