Generic Ontology Design Patterns at Work

Bernd KRIEG-BRÜCKNER [a], Till MOSSAKOWSKI [b] and Fabian NEUHAUS [b]
[a] *Collaborative Research Center EASE, Universität Bremen, Germany, and German Research Center for Artificial Intelligence (DFKI), Bremen, Germany*
[b] *Institute for Intelligent Cooperating Systems, Faculty of Computer Science, Otto-von-Guericke-Universität Magdeburg, Germany*

Abstract. *Generic Ontology Design Patterns*, GODPs, are defined in Generic Dol, an extension of the *Distributed Ontology, Model and Specification Language*, and implemented using the *Heterogeneous Tool Set*. Parameters such as classes, properties, individuals, or whole ontologies may be instantiated with arguments in a host ontology.

The potential of Generic Dol is illustrated with GODPs for an example from the literature, namely Role. We will also discuss how larger GODPs may be composed by instantiating smaller GODPs.

Keywords. ontology design patterns, generic ontologies, generic DOL

1. Introduction

In ontology engineering we may distinguish (at least) three kinds of stakeholders: ontology experts, domain experts and end-users. For each, the kind and level of expertise is quite different: End-users should not be required to have ontology expertise and may have little domain knowledge; domain experts often have little ontology expertise; ontology experts have little specialised domain expertise.

Since many ontology development projects are small and, frequently, not well funded, many of them do not involve a dedicated ontology expert at all or get only very limited input from an ontology expert during the design phase of the ontology. Thus, the majority of the development of an ontology is typically entrusted to domain experts. This may lead to poor design choices and avoidable errors. Furthermore, because of their lack of experience, domain experts may not be able to identify opportunities to reuse existing best practices and ontologies.

Ontology Design Patterns (ODPs) have been proposed for some time as methodology for ontology development, see the early work by [5], the compilation in [8], and the review of the state of the art in [9]. In theory, ODPs provide a solution for the lack of ontology experts: ODPs enable domain ontologists to reuse existing best practices and design decisions and, thus, benefit from the experience of ontology experts who developed the ODPs and lead to a considerable increase in quality of ontologies. However, in practice the adaptation of ODPs as tools for ontology engineers has been slow. In our opinion this is caused by the fact that currently the utilisation of ODPs is cumbersome for ontology developers.

Let us assume an ontology developer intends to reuse a given ODP P for the representation of some domain D during the development of an ontology O; then there are basically two options. Following a *reuse-by-import* stategy, she may import P into O, include the relevant terms for D and link these terms via subclass and subproperty relationships to P [6]. However, this approach clutters the resulting ontology O, because O includes not just terms about D, but all of the generic terms of P. Moreover, an ODP often does not fit into the context of a given ontology completely, but needs to be adapted and some axioms need to be removed. Importation does not provide a means to change P, either one imports it completely into O or not. Alternatively, the ontology developer may follow a *clone-and-modify* approach, i.e. include an axiomatisation of the terms about D that mirrors the axiomatisations of P structurally, and modify it locally. This approach allows more flexibility for adaptions. For example, the XD plugin of the NeOn toolkit [23] provides a GUI for following such an approach.[1] However, manual redesign is error-prone and time-intensive, and the goal of shielding the domain ontologist from the complex design decisions during ontology development is not achieved. Moreover, changes in the original pattern P will not be propagated to O, such that P and O exhibit the typical problems known from code duplication.

In this paper we propose an approach that follows the reuse-by-import approach, but make it more flexible to gain some advantages of the clone-and-modify approach. To this end, we use *Generic Ontology Design Patterns*, GODPs, as a methodology for representing and instantiating ODPs in a way that is adaptable, and allows domain experts (and other users) to safely use ODPs without cluttering their ontologies. The main ideas behind GODPs (Sect. 2) are the following: ODPs are expressed in a dedicated formal, parametrised pattern language that allows (a) the definition of ODPs, (b) specify instantiations of ODPs, (c) extend, modify, and combine ODPs to larger ODPs, and (d) describe the relationships between ODPs. GODPs embody dedicated development operations. They are defined in an extension of the *Distributed Ontology, Model and Specification Language*, DOL, and implemented using the *Heterogeneous Tool Set*, HETS (Sect. 2).

GODPs are patterns in the sense that they contain typed variables. The definition of a GODP involves the specification of parameters that need to be provided for the instantiation of a GODP. Possible parameters include symbols, lists of symbols, but also whole ontologies. The latter enable to express powerful semantic constraints using corresponding axioms; such requirements act like preconditions for instantiations, guaranteeing more consistency and safety.

In this paper we will introduce GODPs by first discussing a toy example in Sect. 2 and afterwards an in-depth discussion of the Role ODP from the literature in Sect. 3. As we will show, there is a straight forward way to embed a ODP like Role in a GODP. In comparison to the 'classical' ODP the corresponding GODP offers the benefit that it is easier to reuse. Further, GODPs enable the nested use of ODPs, which reduces code duplication. Since GODPS are written in an extension of DOL, GDOP developers may utilise the other features of DOL, e.g.,

[1]Of course, it also allows reuse. Both approaches can be useful, see [20].

to explicitly state logical properties of GODPs, represent competency questions, and definitorial extensions. This is significant since it enables quality control on the level of the GODPs as well as their instantiations.

Thus, GODPs provide one tool to support the strategy of ontology reuse by modularisation, which has been proposed by Katsumi and Grüninger [7,12,11]. GODPs shares many objectives with *Parameterised OTTR Templates* with macro expansion by [21]. The differences in the approaches will be analysed in Sect. 4.

The general framework that we use, an extension of DOL, has been developed already in [13,4]. The main contribution of the present paper is the application of this general framework for a specific pattern that has been discussed in the ODP community [14]. Furthermore, we discuss two different strategies for designing and instantiating GODPs: the subsumption and the parametrisation strategy.

2. Generic Ontology Design Patterns in Generic Dol

The *Distributed Ontology, Model and Specification Language*, DOL, an OMG standard [19,16,15], allows the structured definition of ontologies, using import, union, renaming, module extraction, and many more. Thus, DOL is not "yet another ontology language", but a meta-language, which allows to define and manipulate ontologies and networks of ontologies. DOL can be used on top of a variety of languages, in particular OWL 2.

The left column in Fig. 1 contains a small DOL file. After a logic is declared,[2] two different ontologies are specified, namely Driving and DrivingExtended. Note that the expressions in the ontology declaration are not in DOL, but in some ontology language, in this case in OWL Manchester Syntax. (Other ontology languages are available, e.g. FOL-based syntaxes.) DrivingExtended is specified as extension of Driving, which contains an additional domain axiom. The phrase A **then** B in DOL indicates that all definitions in A are visible in B, where A and B are ontologies in OWL (or some other ontology language) or instantiations of GODPs. The semantics of **and** joining two instantiations is union of the two ontologies (i.e. of the corresponding signatures and axioms), which semantically leads to an intersection of their model classes.

We developed Generic Dol [13,4], which extends DOL by a parameterisation mechanism for ontologies. Generics in Generic Dol borrow their semantics from Casl's *generic specification* mechanism [1,18,2]. Generics, first introduced in ADA [10], are not just macros. Their most important aspect is that all parameters are fully typed, and argument types are checked against parameter types.

The syntax for OWL in Generic Dol is presently Manchester Syntax, extended by parameterised names. For example, SimpleRelationGODP in Fig. 1 is a very basic pattern that is defined to utilise three symbols as parameters, namely p (of type object property), D and R (both of type class); their declarations are separated by semicolon ";". The body of the ontology specification (on the right of the "=" symbol) contains an ontology, where p, D and R occur.[3] The GODP

[2]In the following we will omit logic declarations such as **logic** OWL.

[3]Strictly speaking, the body of SimpleRelationGODP is not a legal OWL ontology, since the symbols D and R are not declared. However, they are introduced via the parameter ontologies.

```
logic OWL
ontology Driving =
 Class: Vehicle
 ObjectProperty: drives
  Range: Vehicle

ontology DrivingExtended =
 Driving
then
 Class: Person
 ObjectProperty: drives
  Domain: Person
```

```
pattern SimpleRelationGODP
 [ObjectProperty: p;
  Class: D; Class: R] =
ObjectProperty: p
 Domain: D Range: R
```

```
ontology DrivingPatternInstance =
 SimpleRelationGODP
  [drives; Person; Vehicle]
```

```
Ontology: <DrivingPatternInstance_Exp>
Class: Person
Class: Vehicle
ObjectProperty: drives
 Domain: Person Range: Vehicle
```

Figure 1. DOL Example, and Simple Relation GODP and its application

SimpleRelationGODP may be instantiated by providing suitable arguments. In the definition of the ontology DrivingPatternInstance, the GODP SimpleRelationGODP is instantiated with drives, Person, and Vehicle as arguments.

The *Heterogeneous Tool Set*, HETS [17], is the implementation basis for DOL and GENERIC DOL. HETS is able to compute ontologies that are specified in GENERIC DOL with structuring operations like **then**, **and**, and generics. For this purpose HETS interprets DOL terms and expands the instantiation of generics. This process is called *flattening of an ontology*; the result is a proper OWL ontology. The flattening of the ontologies DrivingExtended and DrivingPatternInstance results in the same ontology; it is shown (in pure Manchester syntax) as DrivingPatternInstance_Exp at the end of Fig. 1.

Parameters in GENERIC DOL are technically ontologies. However, single-symbol parameters are recognised as such; in effect, parameter kinds for single-symbol parameters in OWL are Class, Individual, ObjectProperty, etc. (see [4]). In general, a parameter may be a *complex ontology*, which contains axioms that specify specific abstract properties. An argument ontology must conform to such a parameter ontology, i.e. the required properties must be satisfied. HETS will take care of generating an appropriate proof obligation, if this cannot be deduced automatically. This concept makes GENERIC DOL, and GODPs, extremely powerful to capture semantic preconditions for instantiations (see examples in [13,4]).

Since ontologies DrivingExtended, DrivingPatternInstance and DrivingPatternInstance_Exp are just different representations of the same axioms (namely, declaration of drives, its range, and its domain), what is the benefit of using GENERIC DOL? After all, one can just write the OWL ontology DrivingPatternInstance_Exp directly without the additional complexity. Following the reuse-by-import approach, one benefit of both structuring mechanisms and GODPs is an increased modularity, which enables reusability and avoids code duplication. For example, by dividing the axioms into two modules, it is possible to reuse the axioms in Driving independently from the additional axioms in DrivingExtended. Similarly, it is possible to instantiate SimpleRelationGODP with different parameters to declare a different object property, its domain and range. If at a latter time one

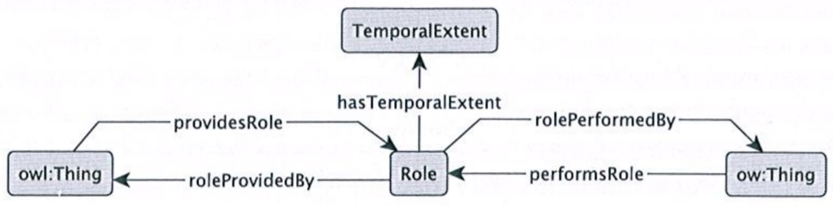

Figure 2. Structure of the Role ODP from [14].

```
Prefix: : <http://www.ontohub.org/godp/>
Ontology: <Role_Original>
Class: Role
 SubClassOf: roleProvidedBy max 1 owl:Thing,
             rolePerformedBy max 1 owl:Thing,
             hasTemporalExtent some TemporalExtent,
             hasTemporalExtent only TemporalExtent
 SubClassOf: roleProvidedBy some owl:Thing
          or rolePerformedBy some owl:Thing
Class: TemporalExtent DisjointWith: Role
ObjectProperty: hasTemporalExtent
ObjectProperty: roleProvidedBy  Domain: Role InverseOf: providesRole
ObjectProperty: rolePerformedBy Domain: Role InverseOf: performsRole
```

Figure 3. Role ODP from [14] in OWL 2 Manchester syntax

needs to change the axioms of Driving or SimpleRelationGODP, these changes are propagated to all ontologies that are based on them. An additional benefit of a GODP is the fact that it hides the internal complexity of its axiomatisation. For the instantiation of a pattern, the user does not need to understand the internal structure of the GODP, he just needs to provide the appropriate parameters.

These benefits are minuscule in the case of a simple ontology like Driving-PatternInstance_Exp in Fig. 1. To illustrate the power of structuring mechanisms and GODPs in Generic Dol, we will discuss an ontology design pattern from the literature below, namely the *Role Pattern*, see Sect. 3.

3. Role Pattern

3.1. The Original Ontology Design Pattern for Role

In [14], the ontology design pattern for Role is introduced, see Fig. 2 for an overview and Fig. 3 for its axiomatisation (which we have converted to OWL 2 Manchester syntax). To summarise, in [14] a role is an entity that has some temporal extent; it is provided by at most one entity and it is performed by at most one entity; and a provider and/or a performer do exist.[4]

[4]For brevity, we have omitted two range declarations that are redundant because the corre-ponding inverse roles have domain declarations.

In [14] ODPs are instantiated by subsumption. For example, for an instantiation of Role for Agent the axioms in Figure 4 are added to the axioms in Figure 3 (modulo translation into Manchester Syntax). The additional axioms in Figure 4 introduce Agent as class and AgentRole as a subclass of Role that is performed by agents. Lastly, it is asserted roles that are performed by agents are agent roles.

```
Class: Agent
Class: AgentRole  SubClassOf: Role, rolePerformedBy only Agent
Class: RolePerformedBySomeAgent
 EquivalentTo: rolePerformedBy some Agent
 SubClassOf: AgentRole
```

Figure 4. Agent defined as instantiation of Role via subsumption in [14]

Note that Agent may be considered as a new ODP, as done in [14]. However, it is an instantiation of the Role pattern in the sense that Figure 4 serves as illustration how Role is to be instantiated not just for Agent, but also in other circumstances. For example, if we would want to instantiate Role for Patient (or some other thematic role), we presumably intend to use similar axioms. Thus, from our perspective the ODP Role does not just consist of the axioms in Figure 3, but also includes the instructions on how to instantiate it in a given context. This instruction is given here by the Agent example.

Illustrating how an ODP is supposed to be instantiated by providing an example is useful, but the technique has some signifiant drawbacks. Firstly, this methodology requires the user to understand the ODP in detail. For example, without familiarity of Role, the necessity for adding the last two axioms in Figure 4 is not apparent. Secondly, the example may not cover important aspects. For example, in the case of Role the example does not cover, how one is supposed to instantiate the ODP for roles that depend on institutions. Hence without detailed understanding of Role, a user will not be able to use Role in order to, for example, represent that Bernd is a professor at the University of Bremen. Given that there are no explicit rules on how to do so, different users may use different approaches to represent institutional roles as instantiations of Role. Thus, if our goal is to enable a team of ontology developers with a moderate level of ontology expertise to use ODPs in a way that is consistent across a large ontology, we need a better methodology than illustration by example.

3.2. Generic Ontology Design Pattern for Role via Subsumption

GODPs uses Generic Dol in order to explicitly encode how a pattern is supposed to be instantiated. Within the reuse-by-import approach, there are two different strategies for the instantiation of a pattern, namely subsumption and parametrisation.[5] Since [14] uses a subsumption strategy, we consider it first. RoleGODPSubsumption in Figure 5 provides an axiomatisation that follows faithfully the Agent example in Figure 4. However, instead of AgentRole and Agent,

[5]In programming language contexts, these are called subtype polymorphism and parametric polymorphism, resp.

the GODP uses two parameters, namely RoleKind and Performer. As we discussed in Sect. 2, a GODP may be instantiated by providing appropriate arguments. In ontology ThematicRoles the GODP RoleGODPSubsumption is instantiated for three different thematic roles. Its definition of Agent and AgentRole is equivalent to Agent as defined in Figure 4. As this example illustrates, one benefit of defining GODPs in Generic Dol is the convenience of being able to instantiate a GODP without needing to repeatedly write axioms as in Figure 4. Indeed one is able to instantiate a GODP without considering the details of its axiomatisation. Furthermore, it is guaranteed that a GODP is always instantiated in the same way and no axioms are forgotten.

```
pattern RoleGODPSubsumption [Class: RoleKind; Class: Performer] =
 Role_Original then
  Class: RoleKind SubClassOf: Role, rolePerformedBy only Performer
  Class: RolePerformedBySome[Performer]  SubClassOf: RoleKind
    EquivalentTo: rolePerformedBy some Performer

ontology ThematicRoles =
    RoleGODPSubsumption[AgentRole;Agent]
and RoleGODPSubsumption[PatientRole;Patient]
and RoleGODPSubsumption[InstrumentRole;Instrument]
```

Figure 5. The Role GODP with via subsumption

Figure 5 illustrates an important feature of Generic Dol, namely parametrised names like RolePerformedBySome[Performer]. Brackets "[" and "]" are used around constituent names. Such a bracketed construct in a name appears at its end, and may contain several constituents, separated by comma "," As a result of substituting parameters by arguments in the expansion of an instantiation, the constituent names are substituted by the corresponding argument names, e.g. RolePerformedBySome[Agent], where Agent is the respective Performer. At the end of this flattening process, parameterised names can be *stratified*: all occurrences of "[" or "," are turned into "_", and (trailing) occurrences of "]" are dropped. Thus, the stratified version of ThematicRoles contains three different classes, namely RolePerformedBySome_Agent, RolePerformedBySome_Patient, and RolePerformedBySome_Instrument.

If we had substituted the parametrised name RolePerformedBySome[Performer] by a 'normal' class name, e.g., RolePerformedBySomePerformer, then ThematicRoles would only contain that class. In this case it would follow that RolePerformedBy some Agent is equivalent with rolePerformedBy some Patient. Hence, provided the other axioms, Agent(a) and performsRole(a, r) would entail that Patient(a)[6] – which is clearly not intended. This example illustrates how the use of parametrised names enables the use of several separate and different instantiations of one GODP within a larger ontology.

[6]If we additionally assume that every agent and every patient performs some agent role or patient role, respectively, then the axioms would even entail that classes Agent and Patient are equivalent.

Parametrised names are a seemingly innocuous feature, but it considerably reduces parameter lists and simplifies instantiations and re-use, since every instantiation automatically generates a new set of (stratified) names, cf. [13,4].

3.3. Generic Ontology Design Pattern for Role via Parametrisation

As we have seen, instantiation of ODPs via subsumption follows the following strategy: one imports the pattern ontology and extends it by adding subclass axioms. As a result the pattern ontology is included in the final result, and all instantiations of the pattern are linked within the subsumption hierarchy. For example, in the ontology ThematicRoles in Figure 5 the classes AgentRole, PatientRole, and InstrumentRole would all be subclasses of Role. In this particular case this might be desirable, but this is not always the case.

For example, the Role ODP as axiomatised in [14] (see Figure 3) includes a recurrent pattern itself. The axioms for providesRole and performsRole are structurally identical: they are partial functions with inverses and their range is restricted to some class. Thus, we may want to identify this pattern as its own GODP, let us call it ScopedPartialFunctionWithInverse[7]. If we would instantiate this GODP with help of the subsumption strategy, the resulting ontology would include an object property ScopedPartialFunctionWithInverse and axioms that establish providesRole and performsRole as subproperties of it. While these axioms are not false, it seems rather undesirable – at least to the authors – to be forced to include these kind of axioms into ones ontology. This is because it clutters the ontology with symbols and axioms that are just technically motivated and are not specific to its domain. We will return to this example below, but for now we just conclude that under some circumstances the subsumption strategy is not ideal, because it requires the importation of the ODP into the final product.

The wholesale importation of the ODP is even more problematic, if the ODP contains some parts that are regarded as optional and are not necessarily useful for every user of the ODP.

The parametrisation strategy for the instantiation of ODPs solves both of these issues. As we have seen in Sect. 2 the application of a parametrised ontology to some arguments leads to a new ontology, in which the parameters are substituted by their arguments. Hence, the generic terms in the GODP are no longer occurring in the resulting ontology. Furthermore, Generic Dol supports optional parameters [4], which may be used to include optional parts of an ontology design pattern.

To see how both of these aspects of Generic Dol are utilised, let us return to the Role example from Figure 2. However, let us assume that the left hand side of the diagram (e.g., the role provider) is optional. In this case, we may define the Role GODP as in Figure 6. This GODP has three parameters, namely Role, Performer, and Provider, the latter is marked as optional with a question mark. Because we need to distinguish between the performers and providers, the occurrences of owl:Thing in Role_Original in Figure 3 have been replaced by the appropriate classes. For the sake of illustrating different aspects of Generic Dol,

[7]The function is scoped in the sense that it does not apply to the whole domain of quantification, but restricted to instances of some class, e.g., Role.

```
pattern RoleGODPParametrisation
  [Class: Role; Class: Performer; ? Class: Provider] =
Class: Role
 SubClassOf: roleProvidedBy[Provider] max 1 Provider,
             rolePerformedBy[Performer] max 1 Performer,
             hasTemporalExtent some TemporalExtent,
             hasTemporalExtent only TemporalExtent
 SubClassOf: roleProvidedBy[Provider] some Provider
          or rolePerformedBy[Performer] some  Performer
Class: TemporalExtent
ObjectProperty: hasTemporalExtent
ObjectProperty: provides[Role]    Range:  Role
ObjectProperty: roleProvidedBy[Provider]  InverseOf: provides[Role]
ObjectProperty: performs[Role]    Range:  Role
ObjectProperty: rolePerformedBy[Performer] InverseOf: performs[Role]
DisjointClasses: Role, TemporalExtent
```

Figure 6. The Role GODP with instantiation via parametrisation

```
  Class: ProfRole
2  SubClassOf: roleProvidedBy_University max 1 University,
              rolePerformedBy_Professor max 1 Professor,
4             hasTemporalExtent some TemporalExtent,
              hasTemporalExtent only TemporalExtent
6  SubClassOf: roleProvidedBy_University some University
           or rolePerformedBy_Professor some Professor
```

Figure 7. Result of RoleGODPParametrisation[Class: ProfRole; Class: Professor; Class: University]

we consider a variant of the role pattern where different instantiations of the pattern do not use the general object properties roleProvidedBy, rolePerformedBy and their inverses, but rather generate specific instantiations of these: we use parametrised names in the object properties (cf. example in Sect. 3.2).

If we apply RoleGODPParametrisation to the arguments ProfRole, Professor and University, the flattened ontology does not contain either of the terms Role, Performer, or Provider because they are substituted (see Figure 7)[8]. Hence, the resulting ontology is more compact than the ontology that would have been as the result of the subsumption strategy. Furthermore, some of the symbols of the GODP have been replaced by instance specific variants (e.g., roleProvidedBy_University) while others are unchanged (e.g., hasTemporalExtent).

In case an optional argument is omitted, all corresponding axioms are deleted. Figure 8 shows the result of applying RoleGODPParametrisation to the arguments MotherRole and Mother with no third argument; thus lines 2 and 6-7 from Figure 7 have no counterpart in Figure 8.

This example illustrates that sometimes care needs to be taken with optional arguments. Some people would expect the axiom *Class: MotherRole SubClassOf:*

[8]For the sake of brevity we include only the first axioms.

```
Class: MotherRole
 SubClassOf: rolePerformedBy_Mother max 1 Mother,
             hasTemporalExtent some TemporalExtent,
             hasTemporalExtent only TemporalExtent
```

Figure 8. Result of RoleGODPParametrisation[Class: MotherRole; Class: Mother;]

rolePerformedBy some Mother to be included among the axioms in Figure 8. However, since missing optional arguments lead to the deletion of axioms, the whole axiom is removed. Removing only the first disjunct would strengthen the axiom. Hence, it is the user's decision whether this strengthening is useful or not.

3.4. Patterns Within Patterns

The presentation of the Role ODP in the diagram in Fig. 2 consists of three parts, namely: 1) the relation of Role to the provider of the role, 2) the relation of Role to the performer of the role, and 3) the relation of Role to TemporalExtent. However, this three-part structure is lost in the axiomatisation of Role in Fig. 3.

An axiomatisation that reflects the modular structure of the Role ODP has several benefits. Firstly, subdividing the axioms into meaningful modules may improve the readability of the ODP. Secondly, it enables the reuse of these modules as ODPs on their own. For example, the axiomatisation of TemporalExtent seems not particular to Role. Thus, it seems sensible to provide this part of the axiomatisation as its own independent GODP that may be reused in other contexts. Finally, the decomposition of Role into smaller GODPs helps to avoid code duplication. As we discussed in Sect. 3.3, the axioms about the provider and the performer are structurally identical, since both introduce partial functions that are restricted to some class as well their inverse. By introducing a GODP called ScopedPartialFunctionWithInverse we avoid maintaining the axioms for this kind of concept twice.

Let us start with the ScopedPartialFunctionWithInverse GODP (see Figure 9). The name of the GODP concisely expresses the involved mathematical properties: a relation with inverse that is a partial function, not necessarily on the whole of owl:Thing, but on a class D (and, thus, scoped). In [14] the ranges of rolePerformedBy and roleProvidedBy are not restricted, see owl:Thing (in Figure 2). Since we intend to enable a distinction between the providers and performers of roles, we include also a range argument R for ScopedPartialFunctionWithInverse.

The GODP hasTemporalExtent asserts that the instances of a given class C have some temporal extent, and that C is disjoint with TemporalExtent. In our running example these axioms are asserted about Role, but they hold for a wide range of classes and, thus, may be reused in other contexts.

With these two building blocks we may now restructure RoleGODPParametrisation along the lines of the diagram in Figure 2. Figure 10 shows the resulting GODP RoleGODPDecomposed. It uses the same parameters as RoleGODPParametrisation. However, its body is significantly shorter, since most axioms have been replaced by applications of GODPs. In line 3 it is asserted that the instances of the class that is provided as argument for the parameter Role have temporal

```
pattern ScopedPartialFunctionWithInverse
  [ObjectProperty: f; Class: D; Class: R; ObjectProperty: finv] =
 ObjectProperty: f  Domain: D  Range: R
 ObjectProperty: finv   InverseOf: f
 Class: D SubClassOf: f max 1 R
```

```
pattern HasTemporalExtent [Class: C] =
 Class: C SubClassOf: hasTemporalExtent some TemporalExtent,
                      hasTemporalExtent only TemporalExtent
 Class: TemporalExtent DisjointWith: C
 ObjectProperty: hasTemporalExtent
```

Figure 9. GODPs ScopedPartialFunctionWithInverse and hasTemporalExtent

```
  pattern RoleGODPDecomposed
2   [Class: Provider; Class: Role; ? Class: Performer] =
   HasTemporalExtent[Role] then
4  ScopedPartialFunctionWithInverse
    [rolePerformedBy[Performer]; Role; Performer; performs[Role]] then
6  ScopedPartialFunctionWithInverse
    [roleProvidedBy[Provider];   Role; Provider;  provides[Role]] then
8  Class: Role SubClassOf: roleProvidedBy[Provider]   some Performer
                        or rolePerformedBy[Performer] some Provider
```

Figure 10. RoleGODPDecomposed consists of a decomposition of the GODP in 3 subparts

extent. Lines 4-5 assert that the variant of rolePerformedBy that is generated depending on the argument for Performer is a scoped partial function (and has an appropriate inverse). Lines 6-7 assert the same for roleProvidedBy. These three applications of GODPs correspond the three parts of the diagram in Figure 2. The last axiom is both about Provider and Performer. Hence, it is not part of either sub-GOPD of RoleGODPDecomposed, but the axiom is added in order to connect the different parts of the GODP.

4. Conclusion, Related and Future Work

In this paper we have illustrated an approach to ontology design patterns that follows the reuse-by-import paradigm, while adding extra flexibility that usually is only available through the clone-and-modify paradigm. We hope that the approach presented here will contribute to the general adoption of ontology design patterns.

As our running example illustrates, our approach is complementary to existing work on ODPs. We started with the axiomatisation of a role pattern from the literature [14] and showed two different strategies for representing it as generic ontology design pattern (GODP). Compared to [14] (and other work on ODPs), the representation of a design pattern as a GODP has the advantage that the rules for the instantiation of a GODP are explicitly encoded within the GODP in Generic Dol. Thus, the instantiation of a GODP may be automatically computed by tools like Hets, which implement the Generic Dol semantics. We have shown that, through the use of GODPs, one can avoid code duplication.

The main difference between the aforementioned different strategies of developing GODPs is the way they are instantiated, namely by subsumption or by parametrisation. As we have illustrated in Sect. 3 with the help of the role pattern, either strategy works. The subsumption strategy is more conservative in the sense that it reuses an existing ODP without any renaming and just adds the necessary information required for automating the instantiation. The parametrised approach is more flexible and powerful, since it utilises the full power of Generic Dol a very flexible variant of reuse-by-import. Both approaches support the nesting of patterns. Further, the strategies are compatible in the sense that a GODP may use a mix of subsumption and parametrisation for its instantiation.

The OTTR approach of *Parameterised Templates* with macro expansion [21, 22] is in many respects very similar to the GODP approach. Apart from ontologies as parameters (cf. [4]), the major contribution of the Generic Dol approach described in this paper is parametrisation supplemented by parameterised names. This is crucial for avoiding unintended name clashes; moreover, *reuse* in distinct contexts is better supported. An extra benefit are the other structuring operations of DOL and the smooth integration with heterogeneous modelling. In particular, this means that GODPs can be used with other ontology languages.

In this paper, we have concentrated on one example from the literature. We expect that salient and well-known patterns in the existing repositories like `ontologydesignpatterns.org`, primarily conceptual or "knowledge" ODPs, will soon be cast into the GODP (or OTTR) framework to make reuse more practical.

All ontologies in this paper can be found online in the Ontohub repository `http://www.ontohub.org/godp`, available both through git and through a web interface.

Acknowledgements

We are very grateful to Serge Autexier, Mihai Codescu, Jens Pelzetter, and Martin Rink for their suggestions and contributions.

References

[1] Egidio Astesiano, Michel Bidoit, Bernd Krieg-Brückner, Hélène Kirchner, Peter D. Mosses, Don Sannella, and Andrzej Tarlecki. CASL - the Common Algebraic Specification Language. *Theoretical Computer Science*, 286:153–196, 2002.

[2] Michel Bidoit and Peter D. Mosses, editors. *CASL User Manual*, volume 2900 of *LNCS*. Springer, Berlin, Heidelberg, 2004.

[3] Eva Blomqvist, Oscar Corcho, Matthew Horridge, Rinke Hoekstra, and David Carral, editors. *8th Workshop on Ontology Design Patterns - WOP 2017*, 2017.

[4] Mihai Codescu, Bernd Krieg-Brückner, and Till Mossakowski. Extensions of Generic DOL for Generic Ontology Design Patterns. In Adrien Barton, Sejla Seppälä, and Daniele Porello, editors, *Proceedings of the Joint Ontology Workshops 2017 Episode V: The Styrian Autumn of Ontology, September 23-25, Graz, Austria*, CEUR Workshop Proceedings. CEUR-WS.org, 2019.

[5] Aldo Gangemi. Ontology design patterns for semantic web content. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *ISWC 2005*, volume 3729 of *LNCS*, pages 262–276. Springer, 2005.

[6] Aldo Gangemi and Valentina Presutti. Ontology design patterns. In *Handbook on ontologies*, pages 221–243. Springer, 2009.

[7] Michael Gruninger, Carmen Chui, and Megan Katsumi. Upper Ontologies in COLORE. In *Proceedings of the Joint Ontology Workshops 2017 Episode 3: The Tyrolean Autumn of Ontology*, Bozen-Bolzano, September 2017. CEUR-WS. `http://ceur-ws.org/Vol-2050/FOUST\_paper\_2.pdf`.

[8] Pascal Hitzler, Aldo Gangemi, Krzysztof Janowicz, Adila Krisnadhi, and Valentina Presutti, editors. *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, volume 25 of *Studies on the Semantic Web*. IOS Press, 2016.

[9] Pascal Hitzler, Aldo Gangemi, Krzysztof Janowicz, Adila A. Krisnadhi, and Valentina Presutti. Towards a Simple but Useful Ontology Design Pattern Representation Language. In Blomqvist et al. [3].

[10] J.D. Ichbiah, B. Krieg-Brückner, A. Wichmann, H.F. Ledgard, J.-C. Heliard, J.R. Abrial, J.P.G. Barnes, and O. Roubine. Preliminary Ada Reference Manual. *ACM SIGPLAN Notices*, 14(6 Part A), 1979.

[11] Megan Katsumi and Michael Grüninger. Choosing Ontologies for Reuse. *Applied Ontology*, 12(3-4):195–221, 2017.

[12] Megan Katsumi and Michael Grüninger. The Metatheory of Ontology Reuse. *Applied Ontology*, 13(3):225–254, 2018.

[13] Bernd Krieg-Brückner and Till Mossakowski. Generic Ontologies and Generic Ontology Design Patterns. In Blomqvist et al. [3].

[14] Adila Krisnadhi. The Role Patterns. In Hitzler et al. [8], pages 313–319.

[15] Till Mossakowski, Mihai Codescu, Fabian Neuhaus, and Oliver Kutz. The Distributed Ontology, Modeling and Specification Language – DOL. In A. Koslow and A. Buchsbaum, editors, *The Road to Universal Logic*, volume I, pages 489–520. Birkhäuser, 2015.

[16] Till Mossakowski, Oliver Kutz, Mihai Codescu, and Christoph Lange. The Distributed Ontology, Modeling and Specification Language. In C. Del Vescovo, T. Hahmann, D. Pearce, and D. Walther, editors, *WoMo 2013*, volume 1081 of *CEUR-WS online proceedings*, 2013.

[17] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The Heterogeneous Tool Set, Hets. In Orna Grumberg and Michael Huth, editors, *TACAS 2007*, volume 4424 of *LNCS*, pages 519–522. Springer, 2007.

[18] Peter D. Mosses, editor. *CASL Reference Manual*, volume 2960 of *LNCS*. Springer, Berlin, Heidelberg, 2004.

[19] Object Management Group. The Distributed Ontology, Modeling, and Specification Language (DOL), 2016. OMG standard available at `omg.org/spec/DOL`. See also `dol-omg.org`.

[20] Valentina Presutti, Eva Blomqvist, Enrico Daga, and Aldo Gangemi. Pattern-based ontology design. In Suárez-Figueroa et al. [23], pages 35–64.

[21] Martin G. Skjæveland, Henrik Forssell, Johan W. Klüwer, Daniel Lupp, Evgenij Thorstensen, and Arild Waaler. Pattern-Based Ontology Design and Instantiation with Reasonable Ontology Templates. In Blomqvist et al. [3].

[22] Martin G. Skjæveland, Daniel P. Lupp, Leif Harald Karlsen, and Henrik Forssell. Practical Ontology Pattern Instantiation, Discovery, and Maintenance with Reasonable Ontology Templates. In Denny Vrandecic et al., editors, *ISWC 2018*, volume 11136 of *Lecture Notes in Computer Science*, pages 477–494. Springer, 2018.

[23] Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, Enrico Motta, and Aldo Gangemi, editors. *Ontology Engineering in a Networked World*. Springer, 2012.