A Good Role Model for Ontologies: Collaborations

Michael Pradel, Jakob Henriksson, and Uwe Aßmann

Fakultät für Informatik, Technische Universität Dresden
michael@binaervarianz.de, {jakob.henriksson|uwe.assmann}@tu-dresden.de

Abstract. Ontologies are today used to annotate web data with machine processable semantics and for domain modeling. As the use of ontologies increases and the ontologies themselves grow larger, the need to construct ontologies in a component-based manner is becoming more and more important. In object-oriented software development, the notions of *roles* and *role modeling* have been known for many years. We argue that role models constitute attractive ontological units—components. Role models, among other things, provide separation of concerns in ontological modeling. This paper introduces roles to ontologies and discusses relevant issues related to transferring these techniques to ontologies. Examples of role models enabling separation of concerns and reuse are provided and discussed.

1 Introduction

Ontology languages are emerging as the de facto standard for capturing semantics on the web. One of the most important ontology languages today is the Web Ontology Language OWL, standardized and recommended by W3C [12]. One issue currently addressed in the research community is how to define reusable ontologies or ontology parts. In more general terms, how to construct an ontology from possibly independently developed components?

OWL natively provides some facilities for reusing ontologies and ontology parts. First, a feature inherited from RDF [7] (upon which OWL is layered) is *linking*—loosely referencing distributed web content and other ontologies using URIs. Second, OWL provides an `owl:imports` construct which syntactically includes the complete referenced ontology into the importing ontology. The linking mechanism is convenient from a modeling perspective, but is semantically not well-defined—there is no guarantee that the referenced ontology or web content exists. Furthermore, the component (usually an ontology class) is small and often hard to detach from the surrounding ontology in a semantically well-defined way. Usually a full ontology import is required since it is unclear which other classes the referenced class depends on. The `owl:imports` construct can only handle complete ontologies and does not allow for partial reuse. This can lead to inconsistencies in the resulting ontology due to conflicting modeling axioms. Overall, OWL seems to be inflexible in the kind of reuse provided, especially regarding the granularity of components.

Existing approaches addressing these issues often refer to *modular ontologies* and, in general terms, aim at enabling the reuse of ontology parts or fragments in a well-defined way (for some work in this direction, see [4–6, 11]). That is, investigate how

only certain parts of an ontology can be reused and deployed elsewhere. While it is interesting work and allows for reuse, we believe that such extracted ontological units fail to provide an intuitive meaning of why those units should constitute *components*—they were not designed as such.

The object-orientated software community has long discussed new ways of modeling software. One interesting result of this research is the notion of *role modeling* [13]. The main argument is that today's class-oriented modeling mixes two related but ultimately different notions: *natural types* and *role types*. Natural types capture the identity of its instances, while a role type describes their interactions. Intuitively, an object cannot discard its natural type without losing its identity while a role type can be changed depending on the current context of the object. *Person* for example, is a natural type while *Parent* is a role type. Parent is a *role* that can be played by persons. A role type thus only models one specific aspect of its related natural types. Related role types can be joined together into a *role model* to capture and separate one specific concern of the modeled whole.

In this paper we introduce role modeling to ontologies. Role modeling can bring several benefits to ontologies and ontological modeling. Roles provide:

- More natural ontological modeling by separating roles from classes
- An appropriate notion and size of reusable ontological components—role models
- Separation of concerns by capturing a single concern in a role model

We believe that role models constitute useful and natural units for component-based ontology engineering. Role models are developed as components and intended to be deployed as such, in contrast to existing approaches aimed at extracting ontological units from ontologies not necessarily designed to be modular. While we argue that modeling with roles is beneficial to ontological modeling and provides a new kind of component not previously considered for ontologies, the transition from object-orientation is not straightforward. The contribution of this paper is the introduction of modeling primitives to support roles in ontologies and a discussion of the main differences for role modeling between ontologies and object-oriented models.[1] The semantics of the new modeling primitives is provided by translation into the assumed underlying ontological formalism of Description Logics (DLs) [3]. That way, existing tools can be reused for modeling with roles. To convince the reader of the usefulness of role models, we demonstrate their use on two examples. The first example shows separation of concerns and the second example demonstrates reuse of role models in different contexts.

The remaining part of the paper is structured as follows. Section 2 introduces roles as used and understood in object-orientation and discusses what the main differences are between models and ontologies. Section 3 introduces role models to ontologies and gives examples of their use. Section 4 discusses related work to component-based ontology modeling and Section 5 concludes the paper and discusses open issues.

[1] When we simply say *model*, we shall mean a model in the object-oriented sense.

2 From Roles in Software Modeling to Ontologies

The OOram software engineering method [13] was the first to introduce roles in object-orientation. The innovative idea was that objects can actually be abstracted in two ways: classifying them according to their inherent properties, and focusing on how they work together with other objects (collaborate). While the use of classes as an object abstraction is a cornerstone in object-oriented modeling, focusing on object collaborations using roles has not been given the attention it deserves (however, for some work addressing these issues, see CaesarJ [1] and ObjectTeams [8]).

There are different views in the object-oriented community [15, 16] on what roles really are. However, some basic concepts seem to be accepted by most authors:

- *Roles and role types.* A role describes the behavior of an object in a certain context. In this context the object is said to play the role. One object may play several roles at a time. A set of roles with similar behavior is abstracted by a role type (just as similar objects are abstracted by a class).
- *Collaborations and role models.* Roles focus on the interaction between objects and consequently never occur in isolation, but rather in collaborations. This leads to a new abstraction not available for classes—the role model. It describes a set of role types that relate to each other and thus as a whole characterizes a common collaboration (a common goal or functionality).
- *Open and bound role types.* Role types are bound to classes by a *plays* relation, e.g. *Person plays Father* (a person can play the role of being a father). However, not all role types of a role model must be bound to a class. Role types not associated with a class are called open and intuitively describe missing parts of a collaboration.

It is important to note that class modeling and role modeling do not replace each other, but are complementary. A purely class-based approach arguably leads to poor modeling by enforcing the representation of role types by classes and thus disregards reuse possibilities based on object collaborations. However, roles cannot replace classes entirely since this would disallow modeling of properties that are not related to a specific context.

Adapting roles for ontology modeling There is currently no consensus on the exact relationship between models and ontologies, although the question is a current and important one (see e.g. [2]). There is however some agreement upon fundamental differences between models and ontologies which will have an impact on transferring the notion of roles from models to ontologies.

One difference is that models often describe something dynamic, for example a system to be implemented. In contrast, ontologies are static entities. Even though an ontology may evolve over time, the entities being modeled do not have the same notion of time. Models often describe systems that are eventually to be executed, while ontologies do not (although some approaches exist that compile ontologies to Java[2]). The dynamism and notion of executability in modeling is closely connected to functionality (or behavior). A collaboration in object-oriented modeling often captures a

[2] See for example, `http://www.aifb.uni-karlsruhe.de/WBS/aeb/ontojava/`.

separate and reusable *functionality*. For example, a realization of depth-first traversal over graph structures may require several collaborating methods in different classes for its implementation. The collection of all the related dependencies between the classes constitutes a collaboration and thus implements this functionality [14]. Because of the non-existence of dynamism and behavior in ontologies, roles and collaborations necessarily capture something different. Instead of describing the *behavior* of an object using the notion of a role, ontological roles describe context-dependent *properties*.

Definition 1 (Ontological roles and role types). *An ontological role describes the properties of an individual in a certain context. A set of roles with similar properties is abstracted by an ontological role type.*

Based on this we define what we consider a role model (collaboration) to be in an ontological setting.

Definition 2 (Ontological collaborations and role models). *An ontological role model describes a set of related ontological role types and as such encapsulates common relationships between ontological roles.*

For example, an ontology may describe the concept *Person*. If *john*, *mary* and *sarah* are said to be persons, but in fact belong to a family, the needed associations may be encoded in a *Family* collaboration describing relationships such as parents having children. The existing *Family* collaboration could then simply be imported and employed to encode that *john* and *mary* are the parents of *sarah*.

Another difference between models and ontologies are their implicit assumptions. In models, classes are assumed to be disjoint, which is, however, not the case for ontologies. This implies that role-playing individuals may belong to classes to which the corresponding role type is not explicitly bound. To avoid unintended role bindings, the ontology engineer explicitly has to constrain them in the ontology.

3 Using Role Models in Ontologies

Class-based modeling, as used in ontologies today, has proven to be successful, but experience in object-orientation has lead to role modeling as a complementary paradigm. This section shows how roles and role models can beneficially be used in ontologies. One of our main motivations is to promote role models as a useful ontological unit—a component—in ontological modeling. We therefore show how role models can be incorporated and reused in class-based ontologies.

The following example is intended to demonstrate how classes can be split into separate concerns where each concern is modeled by employing a different role model. Figure 1 shows parts of a wine ontology modeled with roles. Classes are represented by gray rectangles while white rectangles with rounded corners denote role types. The definition of a role type is specified inside its rectangle (in standard DL syntax). In addition, role types are tagged with the name of their role model, e.g. *(Product)*. Labeled arrows represent binary properties between types.

The ontology in Figure 1 models three natural types (classes): *Wine*, *Winery* and *Food*. In a class-based version of the ontology in Figure 1, the concerns of wine both

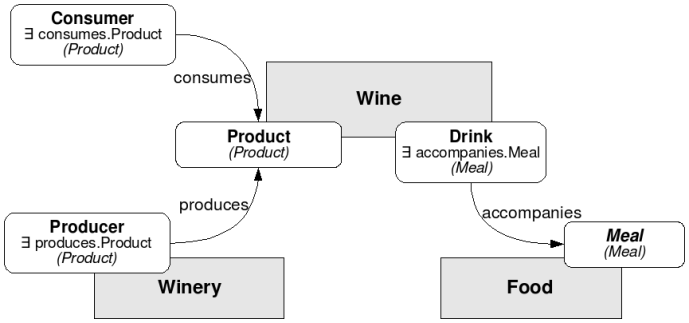

Figure 1. Different concerns of the *Wine* class are separated by the role types *Product* and *Drink*.

being a product and a drink (to be had with a meal) would be intermingled in a single class definition of *Wine*. The most natural way of modeling this would be to state that *Wine* is a subclass of the classes *Product* and *Drink*. However, this would not be ideal since a wine does not always have to be a product. Rather, we would like to express that a wine *can* be seen as a product (in the proper context). This can be expressed using roles where these concerns are instead explicitly separated into the role types *Product* and *Drink*. The motivation from a modeling perspective is that wines are always wines (that is, wine is a natural type). A wine may however be seen differently in different contexts: As a product to be sold, or as a drink being part of a meal. Modeling the role-based ontology from Figure 1 in a more concrete syntax could look like this (based on Manchester OWL syntax [9]):

```
import http://ontology-rolemodels.org/product.rm as p
import http://ontology-rolemodels.org/meal.rm as m

Class: Wine
      Plays: p#Product
      Plays: m#Drink

Class: Winery
      Plays: p#Producer

Class: Food
      Plays: m#Meal
```

The `import` statements import the needed role models and the `Plays` primitive binds roles to classes. The translation of the ontology into standard DL giving the ontology meaning is discussed in Section 3.1.

The above mentioned modeling distinction can also be helpful in other situations. Imagine the existence of an ontology with the classes *Person* and *PolarBear* (naturally) stated to be disjoint. The modeler now wants to introduce the concept of *Parent* and decides that parents are persons. Furthermore, while being focused on polar bears for a while decides that since obviously not all polar bears are parents, the opposite should hold and states that parents are also polar bears. This unfortunate and unintentional mistake makes the class *Parent* unsatisfiable (i.e. it is always empty). A more natural way to solve this problem would be to import a *Family* role model (modeling notions such as parents etc.) and state that *Person*s can play the role of *Parent* and *PolarBear*s

can do the same. Thus, instead of intermingling a *class Parent* with the definitions of *Person* and *PolarBear*, possibly causing inconsistencies, the *role type Parent* cross-cuts the different involved (natural) classes as a separate concern. Doing this will prevent the role type *Parent* from being empty. This example has shown that employing roles can be more natural than using classes to describe non-inherent properties of individuals.

Note that we do not claim that it is not possible to solve the above mentioned modeling problem strictly using classes as is done today. In fact, we very much recognize this fact by giving role-based ontologies a translational semantics to standard DL semantics (see Section 3.1). Instead we argue that modeling with roles is more natural and easier from the perspective of the modeler.

Apart from the rather philosophical distinction between classes and roles described above, roles are important in collaborations. A set of collaborating roles may be joined together in a role model, which may effectively be reused in many different ontologies. Thus, role models provide an interesting *reuse unit* for ontologies.

Figure 2 shows an example of reusability. There are two class-based ontologies, one modeling wines and the other pizzas. Both the concept of *Wine* and *Pizza* in the different ontologies can in certain contexts be considered as products (as one concern). To capture this concern and the relationships the role of being a product has with other roles, for example being a producer, we reuse the *Product* role model introduced in Figure 1.

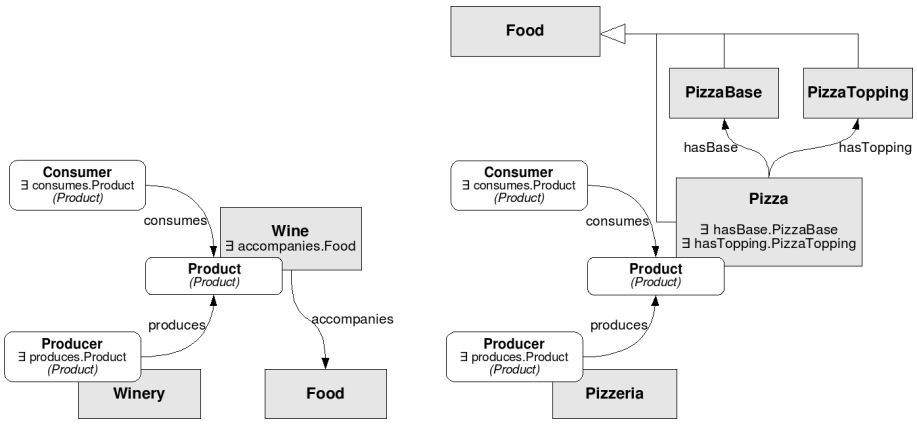

Figure 2. The *Product* role model reused in two different ontologies.

The example shows how a set of related relationships (for example *produces* and *consumes*) can be encapsulated in a role model and reused for different domains. Not only relationships are encapsulated, but also the related role types that act as ranges and domains for the relationships.

As another example we can again consider the previously mentioned *Family* role model where relationships such as *hasChild* and *hasParent* are modeled. This role model may not only be used in an ontology catered to modeling persons. Consider for

instance the same notions being needed in an ontology modeling tree data structures. There, possible relationships between nodes may also be modeled by reusing the same role model. Another example would be an ontology describing operating systems and their processes, new child processes being spawned from parent processes, etc.

After having looked at some examples of ontologies being modeled using role models, we will in the following section discuss their semantics.

3.1 Semantics of Role-Modeled Ontologies

We argue that modeling with roles should be enabled by introducing new ontological modeling primitives. Roles allow modelers to separate concerns in an intuitive manner and provide useful ontological units (components). At the same time, current class-based ontology languages (e.g. OWL) are already very expressive. Thus, we believe that there is no lack in expressiveness, but rather in modeling primitives and reuse. We therefore aim for a translational approach where role-based ontologies may be compiled to standard (DL-based) ontologies. A great advantage is that this permits to reuse existing tools, in particular already well-developed reasoning engines.

A class-based ontology is considered to be a set of DL axioms constructed using class descriptions (or simply classes), property descriptions (or properties), and individuals. For supporting roles, we enhance the syntax with role types and role properties. For the sake of simplicity, we restrict role types to be conjuncts of existential restrictions limited to atomic role types. That is, of the form $\exists p_1.R_1 \sqcap \ldots \sqcap \exists p_n.R_n$, where R_i are role types and p_i are role properties. Role properties simply define their domain and range (both have to be role types). Classes (respectively properties) and role types (respectively role properties) are built from disjoint sets of names. This disjointness corresponds to the underlying difference of natural types and role types.

To support role modeling, we introduce two new axioms. The first axiom expresses that individuals of a class *can* play a role: $R \rhd C$ (*role binding*) binds role type R to class C. The second axiom expresses that some specific individual plays a role: $R(a)$ (*role assertion*), where R is a role type and a an individual. Additionally, we add syntax for ontologies to import role models.

The extended syntax may now be translated to the underlying ontology language by the following algorithm:[3]

1. Make all imported role type definitions available as classes in the ontology.
2. For each role type R used in the ontology:
   (a) Let $\{C_1, \ldots, C_n\}$ be the set of classes to which R is bound ($R \rhd C_i$). Then add the axiom $R \sqsubseteq C_1 \sqcup \ldots \sqcup C_n \sqcup \bot$ to the ontology.
   (b) For each role assertion $R(a)$, make the same assertion available in the resulting ontology, now referring to the class-representative for the role type R.
3. Remove `import` and `Plays` statements.

This translation captures the *can-play* semantics of roles by defining role types as subtypes of classes. It implies that an open role R may not be played by any individual

[3] Role properties and role property assertions are left out here but can be easily integrated into the syntax extensions and the translation algorithm.

since $R \sqsubseteq \bot$ would be added to the ontology (i.e. R is always interpreted as the empty set). The semantics of our role modeling extension is an immediate consequence of the translation by using the standard semantics of DLs.

We will now look at an example of how a role-based ontology is compiled to a standard class-based ontology. The ontology from Figure 1 imports the role models *Product* and *Meal*. The *Product* role model could for example be defined by:[4]

```
RoleModel: http://ontology-rolemodels.org/product.rm
      Role: Producer
            EquivalentTo: produces some Product
      Role: Consumer
            EquivalentTo: consumes some Product
      Role: Product
```

To illustrate the impact of binding one role type to multiple classes, we assume that the *Product* role type is also bound to the class *Food* in Figure 1 (and in the subsequent listing). That is, also foods can be considered products in some contexts. Our translation as defined above results in the following class-based ontology (for the example disregarding the *Meal* role model):

```
Class: Wine
Class: Winery
Class: Food

Class: Producer
      EquivalentTo: produces some Product
      SubClassOf: Winery
Class: Consumer
      EquivalentTo: consumes some Product
      SubClassOf: owl:Nothing
Class: Product
      SubClassOf: Wine or Food
```

The resulting ontology consists of only standard OWL constructs and can thus be used by existing tools such as reasoners. A consequence of this resulting ontology is for example that an individual playing the role of a product has to be either a wine or a food. We can thus single out and study the concern of being a product, but not having to consider in detail what those products are. We could have done the same in a class-based ontology by stating that wines and foods are products, thus using *Product* as a super-class to both *Wine* and *Food*. However, as already mentioned, this would disregard the fact that wines and foods are not always products.

4 Related Work

Modularizing ontologies and finding appropriate ontology reuse units are becoming important issues. Several works address this issue, most having a strong formal foundation. A common property between existing work seems to be the desire to reuse partial ontologies. That is, enable more refined reuse of ontologies by allowing to import and share vocabulary (classes, in some sense *meaning*) rather than axioms (ontologies, that is, syntactical units).

[4] The definitions of the role properties *produces* and *consumes* are left out.

One work in this direction proposes a new import primitive: *semantic import* [11]. Semantic import differs from `owl:imports` (referred to as syntactic import) by allowing to import partial ontologies and by additionally enforcing the existence of any referred external ontologies or ontology elements (classes, properties, individuals) by the notion of *ontology spaces*. The goal in this work is controlled partial reuse.

The work in [5] defines a logical framework for modular integration of ontologies by allowing each ontology to define its *local* and *external* signature (that is, classes, properties etc.). The external signature is assumed to be defined in another ontology. Two distinct restrictions are defined on the usage of the external signatures. The first syntactically disallows certain axioms which are considered harmful, while the second restriction generalized the first by taking semantical issues into consideration. The general goal, apart from a formal framework, is to allow safe merging of ontologies.

The work in [6] also proposes partial reuse of ontologies by allowing to automatically extract modules from ontologies. One interesting requirement put on such an extracted module is that it should describe a well-defined subject matter, that is, be self-contained from a modeling perspective.

In contrast to these works on partial ontology reuse, in particular how to extract or modularize existing ontologies, our work aims at defining a more intuitive ontological unit—an ontological component that was defined as such.

5 Conclusions and Outlook

In this paper we have proposed an ontological unit able to improve modeling and provide a means for reuse—the ontological role model. The concept of roles has its roots in software modeling and we have taken the first steps to transfer this notion to the world of ontologies. Role models provide a view on individuals and their relationships that is different from the abstractions provided by purely class-based approaches. As such, role models provide a reusable abstraction unit for ontologies. Furthermore, due to the translational semantics, the approach is compatible with existing formalisms and tools.

As a next step we aim at integrating role modeling into tools, for example the Protégé ontology editor [10]. This is important since we argue that ontology engineers should treat roles as first class members of their language and distinguish them from classes. Other issues also remain to be further clarified. The semantics of roles may be subject of discussion. Apart from focusing on *can-play* semantics, *must-play* may in some cases be desirable for role bindings. Another issue to clarify is the implication of applying one role model several times in an ontology. One could argue for multiple imports where each import is associated with a unique name space. However, this would disallow to refer to all instances of a certain role type, for instance to all products in an ontology. Finally, further investigations into the implications of the open-world semantics of ontologies relating to role bindings and role assertions should be done.

In conclusion, we argue that role models provide an interesting reuse abstraction for ontologies and that roles should be supported as an ontological primitive.

Acknowledgement

This research has been co-funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net).

References

1. I. Aracic, V. Gasiunas, M. Mezini, and K. Ostermann. *An Overview of CaesarJ*, pages 135–173. Springer Berlin / Heidelberg, 2006.
2. U. Aßmann, S. Zschaler, and G. Wagner. *Ontologies, Meta-Models, and the Model-Driven Paradigm*, pages 249–273. Springer, 2006.
3. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
4. B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Just the right amount: Extracting modules from ontologies. In *Proc. of the Sixteenth International World Wide Web Conference (WWW 2007)*, 2007.
5. B. Cuenca Grau, Y. Kazakov, I. Horrocks, and U. Sattler. A logical framework for modular integration of ontologies. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 298–303, 2007.
6. B. C. Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Modularity and web ontologies. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proceedings of KR2006: the 20th International Conference on Principles of Knowledge Representation and Reasoning, Lake District, UK, June 2–5, 2006*, pages 198–209. AAAI Press, 2006.
7. P. Hayes et al. RDF Semantics. W3C Recommendation, 10 February 2004. Available at http://www.w3.org/TR/rdf-mt/.
8. S. Herrmann. Object teams: Improving modularity for crosscutting collaborations. In *Proc. Net Object Days 2002*, 2002.
9. M. Horridge, N. Drummond, J. Goodwin, A. Rector, R. Stevens, and H. Wang. The manchester owl syntax. *OWL: Experiences and Directions (OWLED)*, November 2006.
10. H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The Protégé OWL plugin: An open development environment for semantic web applications. *Third International Semantic Web Conference (ISWC)*, November 2004.
11. J. Pan, L. Serafini, and Y. Zhao. Semantic import: An approach for partial ontology reuse. In *Proc. of the ISWC2006 Workshop on Modular Ontologies (WoMO)*, 2006.
12. P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL web ontology language semantics and abstract syntax. W3C Recommendation, 10 February 2004. Available at http://www.w3.org/TR/owl-semantics/.
13. T. Reenskaug, P. Wold, and O. Lehne. *Working with Objects, The OOram Software Engineering Method*. Manning Publications Co, 1996.
14. Y. Smaragdakis and D. Batory. Mixin layers: an object-oriented implementation technique for refinements and collaboration-based designs. *ACM Trans. Softw. Eng. Methodol.*, 11(2):215–255, 2002.
15. F. Steimann. On the representation of roles in object-oriented and conceptual modelling. *Data Knowl. Eng.*, 35(1):83–106, 2000.
16. F. Steimann. The role data model revisited. *Roles, an interdisciplinary perspective, AAAI Fall Symposium*, 2005.