Black Cats and Coloured Birds – What do Viewpoint Correspondences Do?

Peter F. Linington
University of Kent Computing Laboratory
Canterbury, Kent, UK
pfl@kent.ac.uk

Abstract

The ODP Reference Model is one of a number of specification frameworks which are based on the definition of a set of viewpoints that are coupled together by the definition of correspondences between terms. Wherever a correspondence is declared, any real world entity that is represented by a term in one viewpoint must also satisfy the requirements placed by the occurrence of the corresponding term in the other viewpoint.

Although this idea represents an intuitively simple and satisfying way of talking about the design of complex systems, the idea of a correspondence is not as simple as it might, at first sight, appear. This paper uses simple examples to illustrate some of the complexities resulting from the coupling of object models and examines the consequences for claims of conformance to the complete system of specifications.

1 Introduction

The ODP architecture [6] [4] [5] defines five viewpoints and a number of correspondences between them. There may, in principle, be up to ten distinct correspondences, because this is the number of possible viewpoint pairings, but designers will not normally need to populate all of them.

When the RM-ODP architecture was first defined, most of the examples considered were in terms of simple correspondences of concepts, terms or names, and not much thought was given to how these correspondences interacted with the basic ideas of object modelling. Since then, however, it has become clear that there are a number of subtleties associated with this structure, which this paper aims to investigate.

To decouple discussion of the nature of correspondences from any prior assumptions about the ODP architecture and what its viewpoints do, we look at an artificial viewpoint structure chosen to illustrate the issues in as simple a way as possible. We assume that all the viewpoints used are expressed in terms of metamodels, and hence object models, in the UML language. The correspondences are also expressed as UML fragments.

The remainder of this paper is organized as follows. In section 2 we give some background to the concept of a viewpoint and its realization in a modelling system. Then in section 3 we introduce the set of simple viewpoint models used as a running example throughout this paper. On this basis, we discuss in section 4 the way viewpoints are composed, and in section 5 investigate the implications this has for conformance. In section 6 we see how variations in the way correspondences are expressed lead to the specification of different worlds. Finally, section 7 draws conclusions and indicates some future directions for the work on viewpoints and correspondences.

2 Background to the viewpoint concept

ODP introduces the concept of a viewpoint to support the separation of concerns in the design process. When working on a large enterprise system it is unrealistic to capture all the necessary constraints and decisions in a single flat specification, or even in a straightforward hierarchical one based on successive refinements.

Structuring the specification into viewpoints gives much more flexibility. For example, an enterprise designer and a middleware engineer can both work in viewpoints where the requirements or constraints of the other party are represented in the simplest, most abstract, form necessary, with links that establish correspondences between the two viewpoints to ensure that all the detail is integrated into a single consistent working solution. The general concept of viewpoints is discussed at length in [3].

In the design of large, long lived systems we need to consider not just the interrelation of viewpoints in a single complete specification, but also the way in which the pieces of specification interact during development. Changes in one viewpoint specification during the design lead to inconsistencies with others, and the correspondences declared can be used to propagate and manage these changes. How this

can be done is discussed in [11] and [12].

A set of viewpoints may not all be expressed in the same way; the different areas of concern may best be addressed with different modelling tools and conventions. The correspondences may link terms that are from different languages, making the unification of the viewpoints a challenging task for the tool provider.

The form of unification depends on the languages used and is outside the scope of this paper, but a detailed study of the formal basis for viewpoint unification mechanisms can be found in [2]. The definition of consistency used there is based on observational equivalence, and a set of viewpoints is considered to be consistent if there exists at least one implementation consistent with all the specifications. However, this paper takes a less formal approach, illustrating some of the issues that arise by the use of simple examples and, as a result, comes to a slightly different view of conformance.

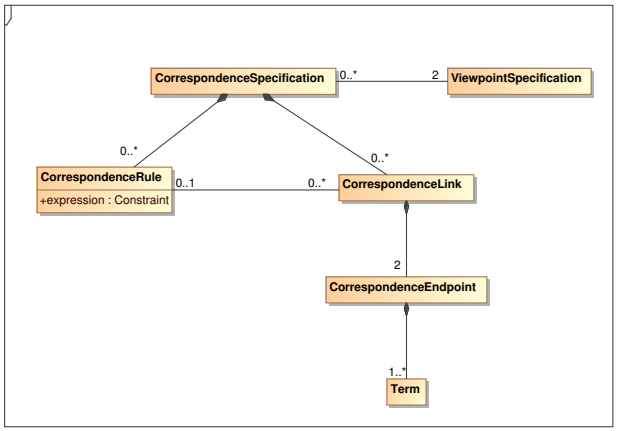

Figure 1. Correspondence specification concepts.

One of the first things to establish is how linkages between models are to be expressed.

In the recently defined UML profile for ODP [7] correspondences are defined as separate specification fragments called *correspondence specifications*. Each of these connects a particular pair of viewpoints and itself consists of a set of *correspondence links* which may themselves have associated *correspondence rules* giving additional constraints. Each link joins two *correspondence endpoints*, and each of these carries a UML tag containing a set of terms in the appropriate viewpoint. The metamodel for this structure is illustrated in figure 1.

It is important to allow each tag to reference a set of terms, rather than a single term, because the correspondence may well associate features in specifications expressed using different levels of abstraction, so that the linkage of terms is not necessarily one to one.

Tags are used here rather than direct associations because doing so maintains the independence between the various model fragments. Indeed, associating textual representations of terms with the tags takes the details of the linkage semantics out of the modelling language used for the fragments, so that issues of name resolution and even of incompatibility of modelling notation between viewpoints do not impact on the preparation of the model fragments or place constraints on the tools used to do this. A particular tool is concerned with just one viewpoint or correspondence at a time. However, correspondence rules must be able to incorporate constraint expressions that are written in terms of the viewpoint linked to, and so the representation of the correspondence rules needs to encapsulate them; such expressions follow the conventions of the viewpoint language, and are opaque terms in the correspondence language. The integration tools that combine viewpoints must, of course, be aware of the format and interpretation of all the elements they need to manipulate.

This paper does not look in detail at the techniques for implementing the integration of viewpoint specifications, but discussion of the kind of tool framework needed can be found in [1], and a transformational approach to correspondences is proposed in [9].

3 A zoological example

As indicated in the introduction, the examples presented here are deliberately artificial, leaving aside technological issues to avoid unnecessary digressions on the basis of commitment to different methodologies or schools of thought on design technique. We want to concentrate here on how the models interact and how different modelling choices have diverging conformance implications.

For this reason, the examples are expressed in simple UML, in term of classes and associations, and do not follow the strict ODP conceptual framework. The correspondences between viewpoints are complex enough, without adding the particular conventions for expressing, for example, composition and binding, defined in [7]. In the interests of readability, we also talk about a *viewpoint* defining something, rather than using the fuller, strictly correct, formulation of a *system specification from a particular viewpoint* defining it.

In this example, we consider a system specification formed by the combination of two domain-specific models, together with two other models concerned with the description of some broad properties of entities. The specific domain is zoological; we consider the description of a simple ecosystem in which a number of species compete.

First, there are two domain specific views. One defines

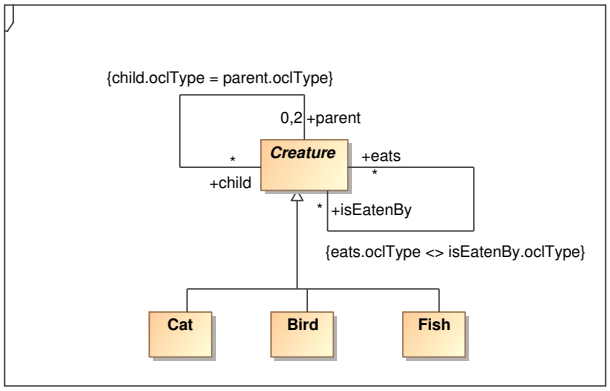

Figure 2. Creatures viewpoint.

three kinds of creature: cats, birds and fish. Relations between individuals are restricted to the description of a family structure and of a part of the food chain. The most basic domain model describes creatures, and is shown in figure 2. One association in this model represents parentage, with an attached OCL [10] constraint that ensures that species breed true; parents and children must be of the same type. Another association represents some link in the food chain, with its own constraint that prohibits cannibalism by ensuring that the types linked by the association are different.

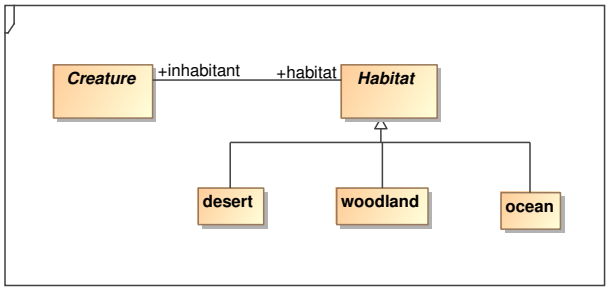

Figure 3. Habitats viewpoint.

The second domain-specific viewpoint describes the association of creatures with the habitats they are adapted to. This view focuses on the properties of the habitat and takes an unspecific view of how creatures are categorized (see figure 3).

The remaining two viewpoints represent generic properties of entities. Firstly, the fact that entities have a colour is represented in a reusable way by defining a separate viewpoint concerned only with colour and colour constraints. To express constraints, the colour of one entity can be constrained by the colour of two other entities by declaring a ternary association called *mingle*, although the nature of the constraint to be applied is not declared in the viewpoint itself. Pairs of sufficiently dissimilar colours are identified by a simple symmetric association called *clashesWith*. The model underlying this viewpoint is illustrated in figure 4.

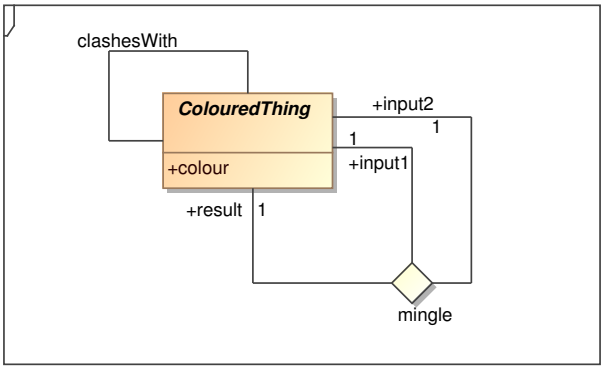

Figure 4. Colour viewpoint.

The second general property is expressed in another small viewpoint expressed by associating a size with each entity. The size attributes of two entities can be related by an ordering relation with endpoints *larger* and *smaller* that represents a partial ordering on the set of entities modelled. This is illustrated in figure 5.

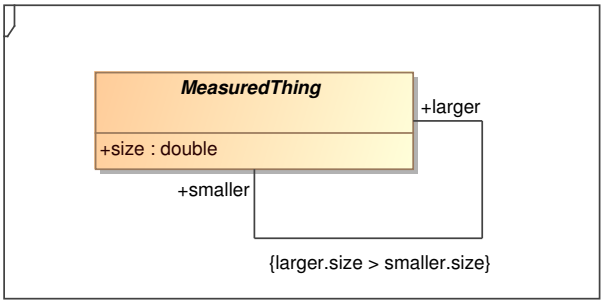

Figure 5. Size viewpoint.

A complete specification might be established by first defining one correspondence between the class Creature in the Creatures viewpoint and the class ColouredThing in the Colour viewpoint, and another between the class Creature and the class MeasuredThing in the Size viewpoint.

The final correspondence that must be declared is between the two domain specific viewpoints. Although both these viewpoints contains a class *Creature*, these are in different models and hence different namespaces, and so an explicit correspondence must be declared between them. So, in summary, we have the following set of constraints equating classes in different viewpoints:

- Creatures.Creature = Habitats.Creature

- Creatures.Creature = Colour.ColouredThing
- Creatures.Creature = Size.MeasuredThing

Without any additional constrains, these specify a model for a Universe of Discourse of creatures which can be cats, birds or fishes, each of which has an unconstrained colour and an unconstrained measurable size, and each of which can live in any habitat.

4 Independence and Composition of Models

The four viewpoint models defined above are independent. That is to say, they are each formed from a separate set of interrelated concepts, but no model makes direct reference to terms in any other model in the collection. To make this clear, we have assumed that each model is formed at the outer level as a distinct package, so that, although names within the scope of each package may have the same textual form, they are initially unrelated.

Correspondences are expressed in a separate model for each pair of viewpoints, as in the UML4ODP standard [7], with each correspondence defining a series of linking items between two endpoints. Each endpoint includes a tag expressing the references by naming something in one of the models being linked by the correspondence; each correspondence will generally include a rule to give some associated constraints on it.

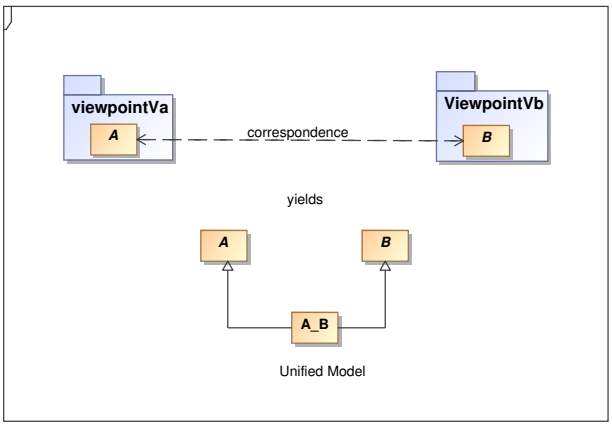

Figure 6. Translating correspondence to shared inheritance model.

By considering the correspondences one at a time, we can, in each case, construct a single model from the two models being linked and the correspondence itself. In one such combination, all the references in the correspondence are satisfied by items in one or other of the viewpoint models being linked, and so the tags can be flattened into common shared classes or associations (so long as the same notations are in use; if not, a more complex linguistic translation process is needed as part of the unification). The result of this process is one model subsuming the previous two models and their correspondence; where some other model had correspondences with both the merged models, these will need to be merged to give just a single correspondence from any other viewpoint to the merged model.

If we wish to unify the class structures, this process can be expressed in terms of a multiple inheritance pattern, so that correspondences between classes can be mapped by the insertion of a single resultant class inheriting from a class in each viewpoint definition. This transformation is illustrated in figure 6.

We can repeat this composition process until we have one large model derived from all the viewpoint models and all the correspondences. If the model composition operations are associative, we can also apply a rewriting to flatten the result at each stage, so that, although an iterative composition would naturally create a deep binary inheritance tree, this can be replaced with a broader but flatter tree in which, in the most complex case from the example (the Creature Class), there is inheritance from four contributors.

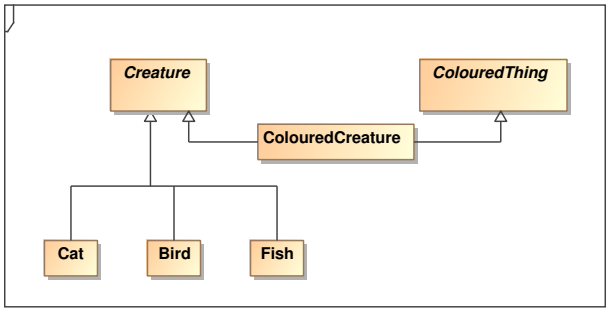

Figure 7. Multiple inheritance is not the right answer.

This works fine for isolated classes, but what about generalizations and associations? First, consider generalization. Creature generalizes Cat, Bird and Fish; if we include these in the picture, the naive approach above would give figure 7, in which cats have no colour, which is not what was intended. In our particular example, we could achieve the right result with a non-commutative merge in which Creature becomes a refinement of the other three corresponding classes; that is to say Creatures.Creature inherits from Habitats.Creature, Colour.ColouredThing and Size.MeasuredThing. However, this only works if just one viewpoint has additional complexity; adding further structure below any of the other classes would then still reintroduce the problem.

So the multiple inheritance approach does not work; we need an approach that distributes the derived properties across the elements of the models being linked, and this will involve a much more radical approach involving rules for rewriting them. Unfortunately, most straightforward rewriting strategies seem to lead to embarrassing complexity. For example, propagating the correspondence down to the leaves of the inheritance tree yields a leaf for each member of the power set of the contributing choices. Alternatively, contracting the tree into a labelling attribute (creatureKind) avoids this problem but at the cost of an explosion of guards if the specializations have extended the base class in any significant way. Further work is needed to identify a more satisfactory target pattern to use.

A similar problem applies for associations. Forming a correspondence between associations leads to implied correspondences between the instances linked by the association, and hence with the classes describing them.

Another problem is that multiple correspondences between classes may lead to unsatisfiable constraints on associations, particularly where fixed multiplicities are specified. An example of a malformed pair of related viewpoints is given in figure 8. Here, creating the correspondences has made it impossible to satisfy the multiplicities in the two viewpoints simultaneously (note that classes in the viewpoints are abstract, so it is only the implied inheriting classes that can be instantiated). This is because the correspondences lead to an implied pair of concrete classes, X_Y and A_B, such that instances of the classes will be required to participate in both a 1:1 and a 1:2 association. Since, by default, the association endpoints in UML form a set, so that instances must be distinct, there is cardinality conflict in the example.

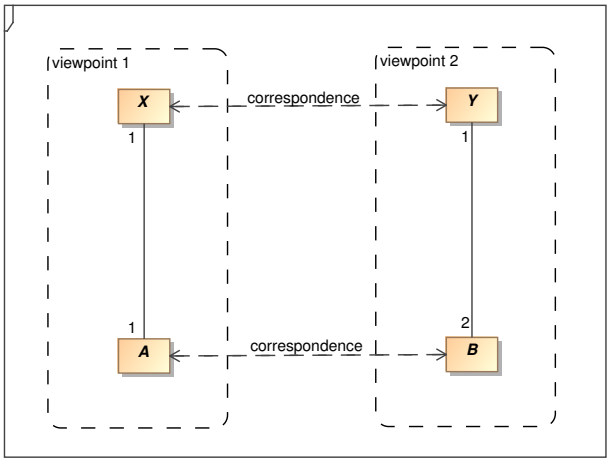

Figure 8. An inconsistent viewpoint correspondence.

Although we are considering UML examples in general, it is worth noticing here that the creation of a correspondence between classes will, in general, imply that the classes in the individual viewpoints are intended to be abstract, and that instances can only have a concrete expression if the constraints from all relevant viewpoints are satisfied. This has implications for the theory of conformance in multi-viewpoint systems, which we examine next.

5 Well-Formedness and Conformance

The previous example illustrated that we need to be concerned with the issues of well-formedness in families of viewpoints, and that this is distinct from the issues of conformance to the set of specifications. Well-formedness is related to absence of contradiction in the specifications, and conformance is related to the absence of contradiction between the specifications and some proposed implementing situation in the real world. We discussed at length the nature of the implementation relationship in [8] and this analysis applies equally but separately to all the viewpoints involved.

We can consider two approaches to conformance to a set of viewpoints. These are:

1. to unify the viewpoints in order to produce a single model and then to check conformance of the observed behaviour of some proposed implementation to that unified model;

2. to check conformance of each viewpoint separately, and while so doing to mark the roles each element of the implementation plays in satisfying the specification; effectively, this corresponds to labelling the implementation elements with terms in the specification that apply to them. Once this process is complete, we can check that the labellings with respect to each of the viewpoints are consistent with the correspondences declared between the viewpoints (note that the labelling may involve asserting that an element definitely plays a role, or definitely does not play a role, or has an undefined status with regard to a particular role labelling).

It was noted above that correspondence tags represented sets of terms. In fact, this is itself problematical, because, although it does allow a correspondence to be made to the components of some fine grain composite that reflects a more abstract element in another viewpoint, the tag content is unstructured; what is needed is not just a set of terms, but the conditions under which they form some pattern equivalent to the implied abstract concept that is being matched.

Thus, for example, suppose an abstract action in one viewpoint corresponds to a specific piece of behaviour, made up of a sequence of finer grain actions, in another viewpoint. The fact that the fine grain actions do indeed

form a sequence is a necessary and significant part of the correspondence. The same actions in a different sequence will not do. Ideally, therefore, the terms in the tag should be associated with roles in a pattern refining the abstract element being matched, and the tag needs to convey all of this information.

6 Constraining with correspondences

At the end of section 3, we gave a minimal set of correspondences that gave our example coherence. Here we want to look at the way the correspondences can be used to place additional constraints, thereby restricting the freedom of choice available in a conforming implementation.

The complete set of correspondences being discussed here is summarized graphically in figure 9. We will build up the full figure in a number of stages illustrating different kinds of correspondence.

6.1 Finer scale class correspondences

Instead of associating Creature with ColouredThing directly, we can introduce specific correspondences for each of the refinements of creature. We can then associate a limitation on the permitted colour with each via a correspondence rule. Now we can have a world populated by, say, black cats, yellow birds and silver fish. Alternatively we can restrict our creatures not just to one colour each, but to restricted ranges of colour. Similarly, we can attach a characteristic colour to specific habitats.

Note that placing these constraints in the correspondence gives a cleaner separation than placing the constraint in either of the viewpoints themselves, since the constraints necessarily refer to terms in both viewpoints. The separation of concerns is not perfect, but restricting the leakage to be within the correspondences gives an effective tool to manage it, since it is then easier to see what terms are referenced in the correspondence rules and so what couplings exist.

6.2 Association correspondences

Correspondences between associations allows different features to be correlated. For example, making the eats/isEatenBy association on Creature correspond to the larger/smaller association on MeasuredThing leads to a requirement that creatures only eat smaller members of a different kind from themselves. Perhaps cats eat sparrows, but eagles eat cats.

It may be more appropriate to make the correspondence between association endpoints than between associations, particularly where the structure of the associations is different in two viewpoints. Thus, for example, to express the way breeding influences colour, we can make one correspondence between endpoint child in Creature and endpoint result in Colour:mingle and others between parent in Creature and the input1 and input2 in Colour:mingle. This gives the framework for expressing some kind of Mendelian constraint on colour. However, the lack of semantics for mingle in the example so far makes this an empty constraint at present. Two further constraints would be needed to complete the picture, one giving the genetic rules, and one the colour combination rules, and these could be imported from the respective viewpoints, assuming the availability of a suitable parameterized constraint structure for rules.

It should also be possible to negate the correspondence. For example, the ColouredThing association *clashesWith* could be defined as being required not to correspond to the Habitat association *inhabits*, because creatures that are not camouflaged get eaten.

Thus, by careful choice of constrained correspondences, quite complex systems can be modelled starting from our simple viewpoints while still maintaining separation of concerns.

6.3 Behavioral correspondences

The example given here concentrates on the static classification structure, but in practical cases it is likely that a very large number of the correspondences will be concerned with actions and aspects of composite behavior. Here, in particular, there will be a need for correspondences between terms at different levels of abstraction, and the difficulties associated with inheritance of behaviour will be conspicuous.

The running example used so far does not have the non-trivial behaviour needed to illustrate the problems, but one might consider, for example, the composition of an application with a reliable transport protocol or a transaction manager. The application might declare an invariant that is true for all successful communication, but the supporting middleware can have hidden recovery actions such that, once the viewpoints are combined, the invariant is violated during what, to the application, had previously been seen as atomic communication actions. Thus the system specification as a whole has changed from having an invariant to having a condition which is true only when a guard for completion of communication is true.

7 Conclusions

This paper has introduced an approach to the interpretation of correspondences based on permissible extensions - that is to say, possible worlds in which all the viewpoints are satisfied by using sets of term labellings that are consistent with the correspondences. This approach provides

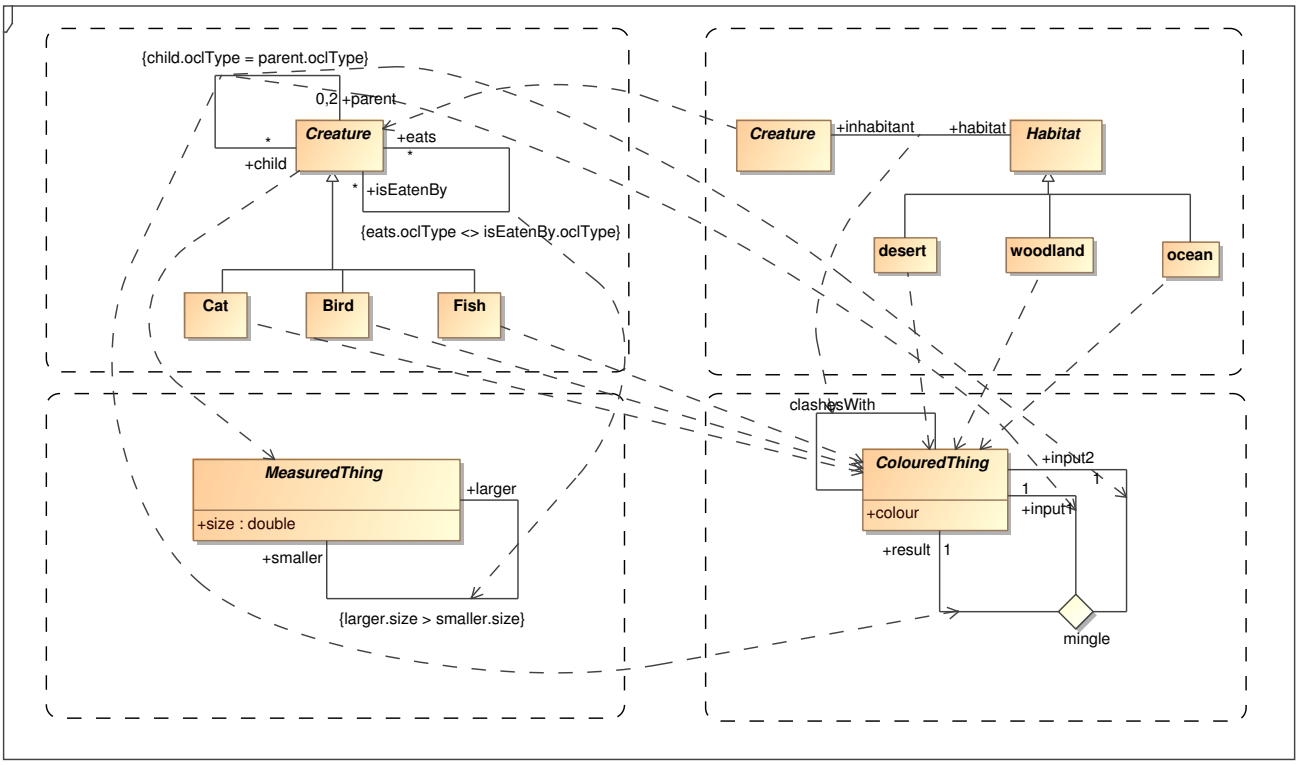

Figure 9. The full set of constraints.

a reference semantics for assessing the correctness of proposed techniques for unifying viewpoint models. As such, it is an approach which is more likely to result in a robust and stable base for viewpoint specification than attempting to provide an operational semantics based on the specification of transformations that manipulate viewpoint models directly to produce a notional complete system model.

However, the next step in the exploration of this approach needs to be a debate amongst experts on the correctness and completeness of larger and more directly relevant examples than the slightly whimsical artificial ones used for illustration here. Hopefully, the discussions presented in this paper will initiate this process and thereby clarify the requirements for tool support for viewpoint management and unification.

Acknowledgements

The author would like to acknowledge the contribution to the initial development of the ideas presented here made by discussion in the ISO SC7 WG19 on Techniques for Specification of IT Systems.

References

[1] D. Akehurst. Proposal for a Model Driven Approach to creating a tool to support the RM-ODP. In *Workshop on ODP for Enterprise Computing (WODPEC 2004), In conjunction with EDOC 2004*, pages 65–68, Monterey, California, Sept. 2004.

[2] M. Bowman, H. Steen, E. A. Boiten, and J. Derrick. A formal framework for viewpoint consistency. *Formal Methods in System Design*, 21(2):111–166, Sept. 2002.

[3] M. Große-Rhode. *Semantic Integration of Heterogeneous Software Specifications (Monographs in Theoretical Computer Science)*. SpringerVerlag, 2004.

[4] *ISO/IEC IS 10746-2, Information Technology - Open Distributed Processing - Reference Model: Foundations*, 1996.

[5] *ISO/IEC IS 10746-3, Information Technology - Open Distributed Processing - Reference Model: Architecture*, 1996.

[6] *ISO/IEC IS 10746-1, Information Technology - Open Distributed Processing - Reference Model: Overview*, 1998.

[7] *ISO/IEC 19793 Information Technology - Open Distributed Processing - Use of UML for ODP system specifications*. Moscow, May 2007. FDIS.

[8] P. F. Linington and W. F. Frank. Specification and implementation in ODP. In J. Cordeiro and H. Kilov, editors, *Proceedings of the 1st Workshop on Open Distributed Processing: Enterprise, Computation, Knowledge, Engineering and Realisation*, pages 69–80, Setubal, Portugal, July 2001. ICEIS Press.

[9] J. R. Romero, N. Moreno, and A. Vallecillo. Modeling ODP correspondences using QVT. In *Model-Driven Enterprise Information Systems, Proceedings of the 2nd International Workshop on Model-Driven Enterprise Information Systems, MDEIS 2006, In conjunction with ICEIS 2006*, pages 15–26, 2006.

[10] J. B. Warmer and A. G. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1999.

[11] N. Yahiaoui, B. Traverson, and N. Levy. Adaptation management in multi-view systems. In *2nd International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT 05)*, pages 99–105, Glasgow, UK, 2005.

[12] N. Yahiaoui, B. Traverson, and N. Levy. A new viewpoint for change management in RM-ODP systems. In *2nd International Workshop on ODP for Enterprise Computing (WODPEC 2005)*, pages 1–6, Enschede, The Netherlands, 2005.