Well-formed Rules for Viewpoint Correspondences Specification

José Raúl Romero
Dept. Informática y Análisis Numérico,
Universidad de Córdoba, Spain
jrromero@uco.es

Antonio Vallecillo
Dept. Lenguajes y Ciencias de la Computación,
Universidad de Málaga, Spain
av@lcc.uma.es

Abstract

Viewpoint modeling is an effective technique for specifying complex software systems in terms of a set of independent viewpoints and correspondences between them. Each viewpoint focuses on a particular aspect of the system, abstracting away from the rest of the concerns. Correspondences specify the relationships between the elements in different views, together with the constraints that guarantee the consistency among these elements. However, most Enterprise Architectural Frameworks, which follow a multi-viewpoint approach, either do not consider the explicit specification of correspondences, or do it in a very simplistic way. In this paper we examine the representation of correspondences in the context of the RM-ODP, identify some of its related issues, and propose some improvements to the way in which correspondences are modeled. In particular, we claim that multi-viewpoint modeling approaches need to specify not only the correspondences between the system views, but also some well-formed rules on such set of correspondence specifications.

1. Introduction

We are witnessing an increasing interest in the Software Engineering community towards the use of models for developing software systems. Models allow to state features and properties of systems accurately, at the right level of abstraction, and without delving into the implementation details. Shifting intellectual property and business logic from source code into models allows organizations to focus on the important aspects of their systems, which have traditionally been blurred by the usage of standard programming languages and underlying technologies. Model-driven engineering (MDE) is an emergent discipline that considers models as first-class entities enabling new possibilities for creating, analyzing, and manipulating systems through various types of tools and languages. Each model usually addresses one concern, and the transformations between models provide a chain that enables the automated implementation of a system from its corresponding models.

Models are specially important in the case of large-scale heterogeneous distributed systems, which are inherently much more complex to design, specify, develop and maintain than classical, homogeneous, centralized systems. One way to cope with such complexity is by dividing the design activity according to several areas of concerns, or **viewpoints**, each one focusing on a specific aspect of the system, as described in IEEE Std. 1471 [11]. Following this standard, current architectural practices for designing open distributed systems define several distinct viewpoints. Examples include the viewpoints described in the "4+1" view model [16], Viewpoints [9], OpenViews [4], Dijkman's framework [5], or the growing plethora of Enterprise Architectural Frameworks (EAF): the Zachman's framework [26], ArchiMate [17], the US Department of Defense Architectural Framework (DoDAF), The Open Group Architectural Framework (TOGAF), the Federal Enterprise Architecture Framework (FEAF), or the Reference Model of Open Distributed Processing (RM-ODP), among others. In particular, the RM-ODP provides five generic and complementary viewpoints on the system and its environment [12]. Each viewpoint addresses a particular concern, and normally uses its own specific (viewpoint) *language*, which is defined in terms of a set of concepts specific that concern, their relationships, and their well-formed rules. A **view** (or *viewpoint specification*, in ODP terms) is a representation of the whole system from the perspective of a viewpoint.

Although separately specified, developed and maintained to simplify reasoning about the complete system specifications, viewpoints are not completely independent: elements in each viewpoint need to be related to elements in the other viewpoints in order to ensure the consistency and completeness of the global specifications. Such relationships are described in terms of **correspondences** [18].

However, most viewpoint modeling approaches to system specification (including the IEEE 1471 standard itself and the majority of the existing EAFs) do not consider correspondences between viewpoints, or assume they are trivially based on name equality between correspondent elements and are implicitly defined. This is a serious problem for large-scale distributed systems in which the viewpoints are indeed separately specified, and in which this simplistic assumption does not hold. The majority of approaches that deal with the problem of inconsistency among viewpoints (see, e.g., [6, 7, 8, 9, 10, 24]) are also based on this oversimplified assumption, which hinders their applicability to many complex systems. Making an analogy with the common 2D representation of 3D figures, this is like drawing independently the three orthographic views of a figure but without defining any correspondence lines between them. As we all know, the consistency and completeness of the specification of the 3D figure cannot be guaranteed unless the appropriate correspondences between the three 2D views are described.

The only EAF that we know allows the explicit definition of correspondences is RM-ODP. This paper reviews the explicit specification of correspondences between viewpoints in the light of that approach, identifies some issues of the existing ISO and ITU-T solution, and proposes a set of improvements that might be required to model correspondences. In particular, we show that the explicit representation of correspondences between elements in the different views is not enough. There are some well-formed rules that constrain this set of correspondences, and that also need to be explicitly specified in any multi-viewpoint modeling approach. We show how to model these well-formed rules, and how they can be supported by tools that check the associated constraints on the sets of correspondences.

2. Viewpoint Correspondences

As mentioned in the introduction, most of the software engineering community efforts have focused on the definition of viewpoints and their corresponding viewpoint languages. However, having a set of independent viewpoints on a system is not enough. These viewpoints should be somehow related, and these relationships made explicit in order to provide a *complete* and *consistent* specification of the system. The questions are: how can it be assured that indeed *one* system is specified? And, how can it be assured that no views impose contradictory requirements? The first problem concerns the conceptual *integration* of viewpoints, while the second one concerns the *consistency* of the viewpoints.

The most general way to address these issues by establishing correspondences between viewpoint elements, which is precisely the approach chosen by the RM-ODP.

In ODP, a **correspondence** is a statement by which some terms or other linguistic constructs in the specification of a viewpoint are associated with (e.g. describe the same entities as) terms or constructs in the specification of a second viewpoint [14].

ODP correspondences do not form part of any one of the five viewpoints, but provide statements that relate the various different viewpoint specifications—expressing their semantic relationships. Hence, we could initially say that a proper ODP system specification consists of a set of viewpoint specifications, together with a set of correspondences between them.

The specifications produced in different ODP viewpoints are each complete statements in their respective viewpoint languages, with their own locally significant names, possibly with different granularity, and so cannot be related without additional information in the form of **correspondence statements** that make clear how elements of different viewpoints are related, and how constraints from different viewpoints apply to particular elements of a single system to determine its overall behavior.

The correspondence statements relate the various different viewpoint specifications, but do not form part of any one of the five basic viewpoints. They fall into two categories [12, Part 3]:

- Some correspondences are required in all ODP specifications; these are called **required correspondences**. If the correspondence is not valid in all instances in which the concepts related occur, the specification is not a valid ODP specification.
- In other cases, there is a requirement that the specifier provides a list of items in two specifications that correspond, but the content of this list is the result of a design choice; these are called **required correspondence statements**.

Examples of *required correspondences* between the ODP computational and engineering viewpoints are [12]:

RC1. Each computational object that is not a binding object corresponds to a set of one or more basic engineering objects (and any channels which connect them). All the basic engineering objects in the set correspond only to that computational object.

RC2. Except where transparencies which replicate objects are involved, each computational interface corresponds exactly to one engineering interface, and that engineering interface corresponds only to that computational interface. The engineering interface is supported by one of the basic engineering objects which corresponds to the computational object supporting the computational interface.

RC3. Where transparencies that replicate objects are involved, each computational interface of the objects being replicated corresponds to a set of engineering interfaces, one for each of the basic engineering objects resulting from the replication. Each of these engineering interfaces corresponds only to the original computational interface.

Similarly, examples of required correspondences between the ODP engineering and technology viewpoints are:

RC4. Each engineering object corresponds to a set of one or more technology objects. The implementable standards for each technology object is dependent on the choice of technology.

RC5. Engineering interfaces correspond to technology interfaces.

RM-ODP only provides required correspondences between the computational and engineering viewpoints, and between the engineering and the technology viewpoints. For the rest of the viewpoints, RM-ODP only states that elements of every viewpoint should be consistent with the specification of the corresponding elements in the rest of the viewpoints, and with the restrictions that apply to them. For instance, the elements of the information viewpoint should conform to the policies of the enterprise viewpoint and, likewise, all enterprise policies should be consistent with the static, dynamic, and invariant schemata defined by the information specification.

The interested reader can consult Part 3 of RM-ODP [12], the Enterprise Language [15] and UML4ODP [13] for the complete set of correspondences between pairs of viewpoints defined by the RM-ODP.

In any case, it is important to distinguish between correspondences between elements of pairs of views, and the constraints on the correspondences themselves, as imposed by the required correspondences. For example, we need to count on mechanisms for specifying a correspondence between two individual objects in two views (a required correspondence statement) and also for specifying and enforcing that *every* object in a view should be related through a correspondence to another object in a second view (as described by a required correspondence). This distinction has not been taken into account in most existing approaches when modeling correspondences, which has somehow limited their expressiveness and analysis capabilities, as we discuss in the next section.

3. Expressing Correspondences: Related Work

Originally, none of the viewpoint-based modeling approaches defined a language or notation to represent correspondences. As mentioned above, relationships between viewpoints were either ignored or briefly mentioned (as it happens, e.g., in the IEEE Std. 1471, the Zachman framework, or most EAFs), or implicitly defined using the names of the related elements (e.g., [6, 7, 8, 9, 10, 24]). The problem is that without explicitly representing correspondences we cannot reason about them, nor properly tackle the integration and consistency issues mentioned above.

Different authors have dealt with the problem of explicitly defining and expressing correspondences between viewpoints, mainly when trying to address the issue of viewpoint consistency checking.

Some of the proposals, e.g. [4, 5, 9, 10, 17, 24], highlight the need to explicitly define and establish these correspondences but do not represent them as independent entities. Rather, they form part of the logical framework they define for checking the consistency of viewpoint specifications.

Other authors explicitly represent correspondences, specially when viewpoint specifications are expressed as UML models, using different alternatives. One interesting possibility is the use of OCL [20] to define relationships between the metamodel elements that represent the appropriate modeling concepts, as suggested by, e.g., Akehurst [1, 2].This approach works very well when the correspondences are defined as constraints between all the instances of certain modeling concepts, e.g., when every computational interface corresponds exactly to one engineering interface (see required correspondence **RC2**). However, there are cases in which correspondences need to be established between particular objects of a specification. The problem is that it is not simple at the metalevel to determine which particular objects should be related. Therefore, it is important that correspondences can be established between specific model elements, too. In addition, not all kinds of correspondences can be modeled as constraints, as it happens for instance when correspondences only express simple relationships (e.g., traces) between the elements

The UML 2.0 language defines *abstraction dependencies*, possibly constrained by OCL statements, as the natural mechanism to represent a relationship that relates two elements or sets of elements that represent the same concept at different levels of abstraction or from different viewpoints [21]. Thus, ODP correspondences between viewpoint specifications (for example, between enterprise objects and information objects, or between enterprise policies and information schemata) can be expressed as UML abstraction dependencies between the corresponding UML model elements. However, as suggested by N. Yahiaoui et al. [25], viewpoint correspondences can also be used for other purposes, e.g., change management in multi-view systems. Change management implies consistent evolution of system specifications: if a view is modified for any reason (e.g., change of some business rules or some QoS requirements), several changes may need to be performed in other views in order to maintain the overall viewpoint consistency. In this context, correspondences act as “binds” that link together the related elements, transforming them if a change in one of them occurs, i.e., propagating the changes to maintain consistency. UML abstraction dependencies show to be insufficient for these purposes. The main reasons are that they cannot store all the required information about the correspondence they represent, and because they can be used to express existence of the correspondence but not to enforce it.

The fact that change propagations can be considered particular cases of model transformations suggests the use of model transformation languages as a good solution to the problem of representing viewpoint correspondences. RM-ODP itself explicitly states that correspondences can be used to define transformations between viewpoint elements to implement consistency checks: “One form of consistency involves a set of correspondence rules to steer a transformation from one language to another. Thus given a specification S_1 in viewpoint language L_1 and specification S_2 in viewpoint language L_2, where S_1 and S_2 both specify the same system, a transformation T can be applied to S_1 resulting in a new specification $T(S_1)$ in viewpoint language L_2 which can be compared directly to S_2 to check, for example, for behavioral compatibility between allegedly equivalent objects or configurations of objects.” [12, Part 3]

The use of *relations* was initially indicated by Akehurst [1, 2] for relating concepts from different viewpoint at the metalevel but not explored any further for relating instances, which is essential for establishing proper correspondences. Dijkman also uses relations and consistency rules in his framework for preserving consistency among viewpoints [5], and shows how they can be effectively used for this purpose. We also explored in a previous work [22] the use of QVT for defining viewpoint correspondences as model transformations. The major limitation of this approach is that it is defined at the metamodel level, which makes it very appropriate for expressing correspondences between all instances of some modeling concepts, e.g., *every* computational object should correspond to one or more engineering objects. However, modeling the correspondence between individual elements of a view does not become very simple and natural. Besides, the current lack of full tool support for the QVT language greatly hampers its application in real settings.

An alternative approach to represent correspondences was finally adopted by ISO/IEC and ITU-T, based on the previous works by Yahiaoui and Traverson [25], in the context of the UML4ODP standardization project [13]. RM-ODP was consciously defined in a notation- and representation-neutral manner to increase its use and flexibility. The need to count with a concrete syntax for RM-ODP motivated ISO/IEC and the ITU-T to start a joint project, launched in 2004, to define a standard (ISO/IEC

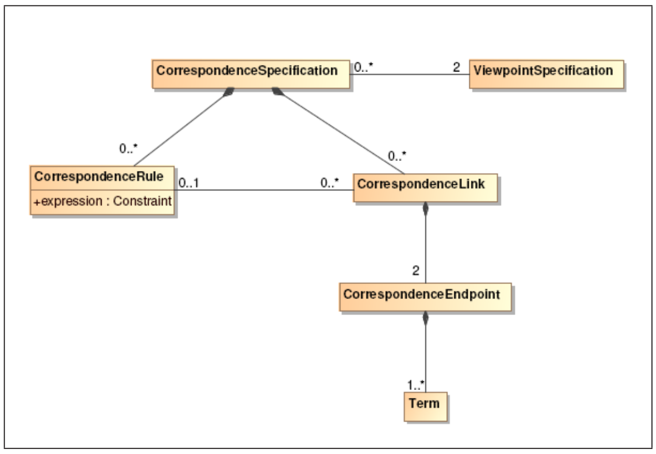


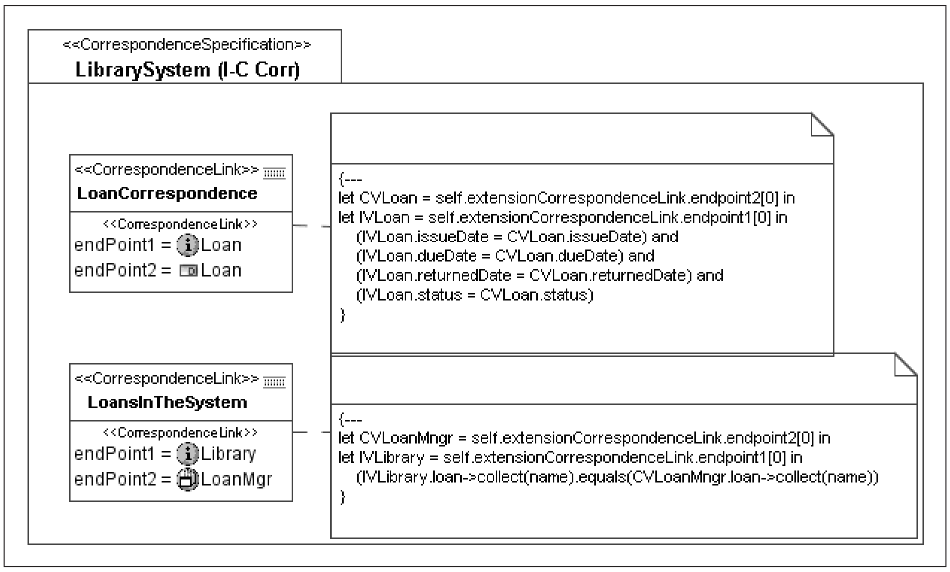
Figure 1. Correspondence metamodel (from [13])

19793 — ITU-T Rec. X.906) for the use of UML for ODP system specifications [13]. This document (usually referred to as UML4ODP) defines a set of UML profiles, one for each viewpoint language, and one to express the correspondences between viewpoints. This correspondence UML profile provides the concrete syntax for a correspondence language, whose abstract syntax is defined by the metamodel depicted in Fig. 1.

In this approach, a *correspondence specification* is composed of a set of correspondence *rules* and a set of correspondence *links*. It describes consistency relationships between terms belonging to two specifications based on different viewpoints. In ODP, a *term* is a linguistic construct which may be used to refer to an entity. The reference may be to any kind of entity including a model of an entity or another linguistic construct. When a correspondence rule and a correspondence link are related, this means that the constraint in the correspondence rule must be enforced by the set of terms referenced by the correspondence link.

In UML4ODP, a correspondence rule is expressed by a constraint that must be enforced by a set of terms belonging to two specifications from different viewpoints. A correspondence link is established between two specifications from different viewpoints. Each end of the correspondence link is called a *correspondence endpoint*, which is composed of terms involved in the consistency relationship.

Two examples of correspondence specifications modeled using this approach are depicted in Fig. 2. It shows first an example of a correspondence between `Loan` information and computational objects: correspondence `LoanCorrespondence` links these two types of objects. The other correspondence establishes that the sets of `Loan` instances in the information view should be consistent with the objects stored by the `LoanMgr` component of the com-



```
{---
let CVLoan = self.extensionCorrespondenceLink.endpoint2[0] in
let IVLoan = self.extensionCorrespondenceLink.endpoint1[0] in
   (IVLoan.issueDate = CVLoan.issueDate) and
   (IVLoan.dueDate = CVLoan.dueDate) and
   (IVLoan.returnedDate = CVLoan.returnedDate) and
   (IVLoan.status = CVLoan.status)
}
```

```
{---
let CVLoanMngr = self.extensionCorrespondenceLink.endpoint2[0] in
let IVLibrary = self.extensionCorrespondenceLink.endpoint1[0] in
   (IVLibrary.loan->collect(name).equals(CVLoanMngr.loan->collect(name))
}
```

Figure 2. Example of a correspondence specification with UML4ODP (from [13])

putational view, which contains the loans stored in the application's database. This is specified by stating that the set of names of the instances of information object loan should coincide with the set of names of the instances stored by `LoanMgr`.

One of the major benefits of this way of modeling correspondences is that it combines the abilities of previous approaches: allowing not only to establish correspondences that express simple relationships (e.g., traces) between multiple elements, but also to express correspondences which need to be modeled as constraints between the related elements (to achieve, e.g., consistency management and synchronization enforcement).

The problem with this approach is that it is very well suited for modeling correspondences between individual elements of the views (i.e., required correspondence statements), but it is not so natural for modeling required correspondences, which define constraints (i.e., rules) that the set of correspondences that comprise the specification should fulfil. This problem is precisely the one that we tackle in this paper.

4. Modeling Viewpoint, Views and Correspondences

In this section we will formulate the specification of viewpoints, views and correspondences from a metamodeling approach. Metamodeling is intended as a common technique for defining the abstract syntax of models and the interrelationships between model elements. A model is an abstraction of a system from a given perspective, and a metamodel is yet another abstraction, describing properties of the model itself. A model is said to *conform to* its *metamodel* in an analogous way a program conforms to the programming language in which it is written, or a XML document conforms to an XML schema [3, 23].

In this context, a view is an abstraction of a software system, highlighting properties of the model itself. Correspondence specification between viewpoints is yet an another abstraction, tracing relations between viewpoint elements. Thus, the natural way to define viewpoint languages in this scenario is by using metamodels, and then views (i.e., viewpoint specifications) are just models that conform to these metamodels.

The way to specify correspondences is by using models, too, which conform to the appropriate metamodels. Such correspondence metamodels can be defined either ad-hoc (e.g., the one defined in UML4ODP, and shown in Fig. 1), using OCL constraints [2], or using a Model Transformation language to define viewpoint correspondences as model transformations (e.g., using the QVT metamodel [19]).

We can then formulate from a modeling perspective the initial approaches (including, e.g., the Zachman framework, IEEE Std. 1471 and most EAFs: TOGAF, DoDAF, FEAF, etc.) to define a multi-viewpoint specification of a system as follows:

Definition 1 (Initial) *A* System Specification *consists of a set of views $V = \{V_1, \ldots, V_n\}$. Each view V_i is a model that conforms to a metamodel $\mathcal{M}_i$ (the viewpoint language).*

Although the relationships (i.e., correspondences) between the views are mentioned, these approaches neither define precise concepts and mechanisms for specifying correspondences, nor notations for modeling them. In this sense, correspondences are *implicitly* defined in these approaches.

Other approaches, such as those mentioned in Section 3, propose the explicit specification of correspondences between viewpoints:

Definition 2 (With explicit correspondences) *A* System Specification *consists of a set of views $V = \{V_1, \ldots, V_n\}$ and a set of correspondences $C = \{C_{(1,2)}, C_{(1,3)}, \ldots, C_{(n-1,n)}\}$ between the views. Each view V_i is a model that conforms to a metamodel $\mathcal{M}_i$ (the viewpoint language). Correspondences are also models, and each $C_{(i,j)}$ conforms to a correspondence metamodel $\mathcal{C}$.* [1]

However, one of the problems of these approaches is that they define correspondences either (a) at model level (e.g., UML4ODP); or (b) at metamodel level (e.g., OCL or QVT). In the first case, correspondences are defined as models that link elements of the views. In the second case, correspondences are defined between metamodel elements. This is why the first ones are more appropriate for modeling correspondences between individual elements, and the second ones are more apt for modeling correspondences that happen between all the instances of a given metamodel element. However, none of the two approaches allow a natural representation of both kinds of correspondences, as mentioned above in Section 3, and neither of them allow the specification of the required correspondences (which describe the well-formed rules that the set of correspondences between views elements should obey).

This justifies the following definition of multi-viewpoint system specification:

Definition 3 (With well-formed correspondences) *A* System Specification *consists of a set of views $V = \{V_1, \ldots, V_n\}$, a set of correspondences $C = \{C_{(1,2)}, C_{(1,3)}, \ldots, C_{(n-1,n)}\}$ between the views, and a set of rules $R = \{r_1, \ldots, r_k\}$ that describe the constraints that the correspondences of C should fulfil in order for a specification to be well-formed. Each view V_i is a model that conforms to a metamodel $\mathcal{M}_i$ (the viewpoint language). Correspondences are also models, and $C_{(i,j)}$ conforms to a correspondence metamodel $\mathcal{C}$. Rules are expressed as constraints on the correspondence elements, using any constraint language (e.g., OCL).*

Thus, our proposal is to use the UML4ODP correspondence metamodel and profile for modeling the required correspondence statements between individual model elements, and then OCL constraints to describe the rules on the correspondences (i.e., the required correspondences).

[1] In this paper we assume that all correspondences conform to the same metamodel $\mathcal{C}$, instead of having independent metamodels $\mathcal{C}_{(i,j)}$ for each correspondence. We believe this is a realistic restriction from a practical point of view.

5. Expressing well-formed correspondences

As mentioned before, we need to declare well-formed rules that establish valid constraints between all the element instances involved in the multi-view specification. These rules permit declaring required correspondences by customizing the correspondence metamodel to each specific system, imposing constraints on its instances. In general, most ODP specifications will need to apply just a set of some predefined rules, like those defined by RM-ODP [12], as previously explained in Section 2, and a set of designer-defined statements for the ODP system being modeled.

For example, let us consider the required correspondence **RC1** described in Section 2, which states that there should be a correspondence between each computational object that is not a binding object, and a non-empty set of basic engineering objects; and that all the basic engineering objects in that set should be related only to that computational object.

This rule should be defined using some constraint language. OCL seems to be a very suitable alternative in this case, since this language is also aligned to MOF 2.0, i.e., the language used to define the correspondence metamodel, and it also has some initial tool support. Thus, in OCL this constraint could be expressed as follows.

```
-- Required Correspondence RC1
context CorrespondenceSpecification inv:
  let CVOBJECTS =
    self.viewpointSpecification
    ->select(o : CV_Metamodel::CV_Object |
      not oclIsTypeOf(CV_Metamodel::BindingObject))
  in
  let NVOBJECTS =
    self.viewpointSpecification
    ->select(n : NV_Metamodel::BEO)
  in
  let CORRESPONDENCES =
    CorrespondenceLink->allInstances()->select(
       (correspondenceEndpoint[0]->size()=1 and
        correspondenceEndpoint[0].
          oclIsTypeOf(CV_Metamodel::CV_Object) and
        correspondenceEndpoint[1].notEmpty() and
        correspondenceEndpoint[1]->forAll(
          oclIsTypeOf(NV_Metamodel::BEO)))
     or
       (correspondenceEndpoint[1]->size()=1 and
        correspondenceEndpoint[1].
          oclIsTypeOf(CV_Metamodel::CV_Object) and
        correspondenceEndpoint[0].notEmpty() and
        correspondenceEndpoint[0]->forAll(
          oclIsTypeOf(NV_Metamodel::BEO))))
  in
  CVOBJECTS->size()=CORRESPONDENCES->size() and
  NVOBJECTS->forAll(n |
    CVOBJECTS->exists(o | isRelated(o,n)) and
    CVOBJECTS->forAll(o1,o2 |
            isRelated(o1,n) and isRelated(o2,n)
        implies o1 = o2))
    )
```

In that OCL expression we make use of an auxiliary function that determines if two model elements are related by any of the existing correspondences in the specification:

```
context CorrespondenceSpecification::isRelated
 (o1 : ModelElement, o2 : ModelElement) : Boolean
body
CorrespondenceLink->allInstances()->exists(c |
 (c.correspondenceEndpoint[0].term->includes(o1)
  and
  c.correspondenceEndpoint[1].term->includes(o2))
 or
 (c.correspondenceEndpoint[1].term->includes(o1)
  and
  c.correspondenceEndpoint[0].term->includes(o2)))
```

Please notice how the OCL constraint that specifies required correspondence **RC1** is declared at the highest level of the system specification, because it involves elements from two viewpoints as well as correspondence statements.

Other correspondences (e.g., **RC2** to **RC5**) can be specified in a similar way.

Finally, it is important to note that we have not considered here the problem of intra-viewpoint consistency, since we assume that all views are well-formed and conform to their respective metamodels, including the correspondence specification.

6. Conclusion and Future Work

This paper addresses the problem of providing precise specifications of correspondences between viewpoints, as a first step to tackling the problem of inter-viewpoint consistency. We have discussed how current approaches to multi-viewpoint specifications of software systems either do not consider the explicit specification of correspondences or, in the cases that take them into account, do not consider the need to explicitly specify well-formed rules on the set of correspondence specifications.

From our perspective, we need to distinguish between two different kinds of constraints on the system elements: $a)$ a set of possibly constrained *correspondences* to specify the relationship between specific elements declared in different views; and $b)$ a set of *well-formed rules* to describe the constraints that the set of correspondences should fulfil.

We have shown one example of our proposed approach in the context of the RM-ODP, although it can also be used in the rest of the multi-viewpoint approaches currently available, and in particular in existing EAFs: TOGAF, DoDAF, FEAF, etc. Furthermore, we expect that it can be considered in the revision process of IEEE Std. 1471, currently undertaken by ISO and IEEE and that will produce the new International Standard ISO/IEC 42010. As we have discussed in the paper, both correspondences between viewpoints and also their well-formed rules should be explicitly specified in any multi-viewpoint specification.

One of this benefits of our approach is that it can be supported by tools: the only requirement is to count on an OCL engine (or an engine for any other supported constraint language), capable of checking that both the constraints defined for each correspondence specification and the constraints that define the well-formed rules on the set of correspondences. As part of our future plans, we expect to implement this approach in the plugin we are developing with MagicDraw for ODP system specifications. This plugin will not only allow the use of the UML profile for ODP, as stated in [13], for the specification of the five ODP views and their respective correspondences, but also it will allow to validate both such views in order the user to ensure that they conform to their respective metamodels (intra-viewpoint consistency) and to check that any correspondence is properly fulfilled by the specification.

Acknowledgements The authors would like to thank the anonymous referees for their insightful comments and suggestions. This work has been partially supported by Spanish Research Projects TIN2005-09405-C02-01, P07-TIC-03184 and PET2006-0682-00.

References

[1] D. Akehurst, S. Kent, and O. Patrascoiu. A relational approach to defining and implementing transformations between metamodels. *Software and Systems Modeling (SoSyM)*, 2(4):215–239, 2003.

[2] D. H. Akehurst. Proposal for a model driven approach to creating a tool to support the RM-ODP. In *Proc. of WODPEC 2004*, pages 65–68, Monterey, California, Sept. 2004.

[3] J. Bézivin. On the unification power of models. *Software and Systems Modeling (SoSyM)*, 4(2):171–188, 2005.

[4] E. A. Boiten, H. Bowman, J. Derrick, P. Linington, and M. W. Steen. Viewpoint consistency in ODP. *Computer Networks*, 34(3):503–537, August 2000.

[5] R. M. Dijkman, D. A. Quartel, and M. J. van Sinderen. Consistency in multi-viewpoint design of enterprise information systems. *Information and Software Technology*, 50(7-8):737–752, June 2008.

[6] S. Easterbrook and B. Nuseibeh. Using viewpoints for inconsistency management. *Software Engineering Journal*, pages 31–43, Jan. 1996.

[7] A. Egyed. Instant consistency checking for the UML. In *Proc. of the 28th International Conference on Software Engineering (ICSE '06)*, pages 381–390, New York, NY, USA, 2006. ACM.

[8] A. Egyed. Fixing inconsistencies in UML design models. In *Proc. of the 29th International Conference on Software Engineering (ICSE'07)*, pages 292–301, Washington, DC, USA, 2007. IEEE Computer Society.

[9] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple prespectives in systems development. *International*

Journal on Software Engineering and Knowledge Engineering, 2(1):31–58, 1992.

[10] M. Große-Rhode. *Semantic Integration of Heterogeneous Software Specifications*. Springer-Verlag, Berlin, 2004.

[11] IEEE. *Recommended Practice for Architectural Description of Software-Intensive Systems*. New York, USA, 2000. IEEE Std. 1471.

[12] ISO/IEC. *RM-ODP. Reference Model for Open Distributed Processing*. ISO and ITU-T, Geneva, Switzerland, 1997. ISO/IEC 10746, ITU-T Rec. X.901-X.904.

[13] ISO/IEC. *Information technology – Open distributed processing – Use of UML for ODP system specifications*. ISO and ITU-T, Geneva, Switzerland, 2008. ISO/IEC FDIS 19793, ITU-T X.906.

[14] ISO/IEC. *RM-ODP. Reference Model for Open Distributed Processing – Amendment to Parts 2 and 3*. ISO and ITU-T, Geneva, Switzerland, 2008.

[15] ISO/IEC 15414, ITU-T Rec. X.911. *Information technology – Open distributed processing – Reference model – Enterprise language*. ISO/IEC and ITU-T, 2006.

[16] P. Kruchten. Architectural blueprints — The “4+1” view model of software architecture. *IEEE Software*, 12(6):42–50, Nov. 1995.

[17] M. Lankhorst et al. *Enterprise Architecture at Work*. Springer, 2005.

[18] P. Linington. Black Cats and Coloured Birds What do Viewpoint Correspondences Do? In *Proc. of the 4th International Workshop on ODP and Enterprise Computing (WODPEC 2007)*, Maryland, USA, Oct. 2007. IEEE Digital Library.

[19] OMG. *MOF QVT Final Adopted Specification*. Object Management Group, Nov. 2005. OMG doc. ptc/05-11-01.

[20] OMG. *Object Constraint Language (OCL) 2.0*. OMG, Needham (MA), USA, May 2006. OMG doc. ptc/06-05-01.

[21] OMG. *Unified Modeling Language 2.1.1 Superstructure Specification*. OMG, Needham (MA), USA, Feb. 2007. OMG doc. formal/07-02-05.

[22] J. R. Romero, N. Moreno, and A. Vallecillo. Modeling ODP Correspondences using QVT. In *Proc. of MDEIS’06*, pages 15–26, 2006.

[23] B. Selic. The Pragmatics of Model-driven Development. *IEEE Software*, 20(5):19–25, 2003.

[24] V. D. Straeten, Simmonds, and Mens. Detecting inconsistencies between UML models using description logic, 2003.

[25] N. Yahiaoui, B. Traverson, and N. Levy. Adaptation management in multi-view systems. In *Proc. of WCAT’05*, pages 99–105, Glasgow, Scotland, UK, July 2005.

[26] J. A. Zachman. *The Zachman Framework: A Primer for Enterprise Engineering and Manufacturing*. Zachman International, La Cañada (CA), USA, 1997. `http://www.zifa.com`.