MODELING ODP CORRESPONDENCES USING QVT

José Raúl Romero
Universidad de Córdoba
Email: jrromero@uco.es

Nathalie Moreno, Antonio Vallecillo
Universidad de Málaga
Email: {vergara,av}@lcc.uma.es

Keywords: Viewpoint modeling, Model transformations, Open Distributed Processing, ODP correspondences

Abstract: Viewpoint modeling is currently seen as an effective technique for specifying complex software systems. However, having a set of independent viewpoints on a system is not enough. These viewpoints should be related, and these relationships made explicit in order to count with a set of complete and consistent specifications. RM-ODP defines five complementary viewpoints for the specification of open distributed systems, and establishes correspondences between viewpoint elements. ODP correspondences provide statements that relate the various different viewpoint specifications, expressing their semantic relationships. However, ODP does not provide an exhaustive set of correspondences between viewpoints, nor defines any language or notation to represent such correspondences. In this paper we informally explore the use of MOF QVT for representing ODP correspondences in the context of ISO/IEC 19793, i.e., when the ODP viewpoint specifications of a system are represented as UML models. We initially show that QVT can be expressive enough to represent them, and discuss some of the issues that we have found when modeling ODP correspondences with QVT relations.

1 Introduction

Viewpoint modeling is gaining recognition as an effective approach for dealing with the inherent complexity of the design of large distributed systems. It comprises two major elements: model-driven development (MDD) on the one hand, and viewpoints on the other. The first one uses models as the key elements to direct the course of understanding, design, construction, deployment, operation, maintenance and evolution of systems. Models allow to state features and properties of systems accurately, at the right level of abstraction, and without delving into the implementation details—or even without giving a solution of how these properties can be achieved (Große-Rhode, 2004). Viewpoints divide the system design according to several areas of concerns, and have been adopted by the majority of current software architectural practices, as described in IEEE Std. 1471 (IEEE Std. 1471, 2000).

The Reference Model of Open Distributed Processing (RM-ODP) framework (ISO/IEC 10746-1 to 10746-4, ITU-T X.901 to X.904, 1997) provides five generic and complementary viewpoints on the system and its environment: *enterprise*, *information*, *computational*, *engineering* and *technology* viewpoints. They allow different stakeholders to observe the system from different perspectives (Linington, 1995). In addition, five viewpoint languages define the concepts and rules for specifying ODP systems from these viewpoints.

ODP viewpoint languages are abstract, in the sense that the RM-ODP defines their concepts and structuring rules, but independently from any notation or concrete syntax to represent them. This allows focusing on the modeling concepts themselves rather than on notational issues, and also allows the use of different notations, depending on the particular needs and on the appropriateness of the specific notation, e.g., Z for the information viewpoint, or Lotos for the computational viewpoint. The RM-ODP architectural semantics (ISO/IEC 10746-4, ITU-T Rec. X.904, 1998) deals with the representation of ODP concepts in different languages. Although ODP does not prescribe the choice of specification language to be adopted with the particular viewpoints, it does advocate that the chosen languages should be formal (Bowman et al., 1995). However, this notation-independence may also bring along some limitations, e.g., it may hinder the development of ODP tools. The need to count with precise notations for expressing ODP specifications, and to develop ODP tools, motivated ISO/IEC and ITU-T to launch a joint project in 2004 which aims to define the use of UML for ODP system specifications (ISO/IEC CD 19793, ITU-T Rec. X.906, 2005). This new initiative (hereinafter called UML4ODP) is expected to allow the development of tools for writing and analyzing ODP specifications, and to make use of the latest MDD practices for designing and implementing ODP systems. UML4ODP defines a metamodel for each ODP viewpoint language, and a set of UML Profiles for representing them. In this way, the ODP viewpoint specifications are expressed as a set of UML models of the system. This initiative introduces very interesting benefits: ODP modelers can use the UML notation for expressing their ODP specifications in a standard graphical way; UML modelers can use the RM-ODP concepts and mechanisms to structure their UML system specifications; and modeling tool suppliers can develop UML-based tools that are capable of expressing ODP system specifications.

So far, most of the ODP community efforts have focused on the definition of the five 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.

RM-ODP tries to address these issues by establishing correspondences between viewpoint elements. 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, a proper ODP system specification consists of a set of viewpoint specifications, together with a set of correspondences between them.

ODP does not provide however an exhaustive set of correspondences between viewpoints (ODP is silent about many of them), nor defines any language or notation to represent correspondences. But without explicitly representing them we cannot reason about them, nor properly tackle the integration and consistency issues mentioned above.

In this paper we explore the use of MOF QVT (OMG, 2005) for representing ODP correspondences in the context of UML4ODP, i.e., when the ODP viewpoint specifications of a system are represented as UML models. We show that QVT seems to be expressive enough to represent them, and discuss some of the issues that we have found when modeling ODP correspondences with QVT.

The structure of this paper is as follows. First, Section 2 provides a brief introduction to ODP, describes in detail the correspondences defined by ODP, and also discusses some previous proposals for representing ODP correspondences. Section 3 provides a short introduction to QVT. Then, Section 4 presents our initial proposal, describing how to represent ODP correspondences with QVT. Section 5 discusses some the issues that we have found during our work. Finally, Section 6 draws some conclusions and outlines some future research activities.

2 ODP

2.1 Introduction to ODP

RM-ODP is a reference model that aims at integrating a wide range of present and future ODP standards for distributed systems, maintaining consistency among them. The reference model provides the coordination framework for ODP standards, and offers a conceptual framework and an architecture that integrates aspects related to the distribution, interoperation and portability of software systems—in such a way that hardware heterogeneity, operating systems, networks, programming languages, databases and management systems are transparent to the user. In this sense, RM-ODP manages complexity through a "separation of concerns", addressing specific problems from different points of view.

In ODP terms, a *viewpoint* (on a system) is an abstraction that yields a specification of the whole system related to a particular set of concerns. ODP defines five viewpoints, covering all the domains of architectural design. These five viewpoints are:

- the **enterprise** viewpoint (EV), which is concerned with the purpose, scope and policies governing the activities of the specified system within the organization of which it is a part;
- the **information** viewpoint (IV), which is concerned with the kinds of information handled by the system and the constraints on the use and interpretation of that information;
- the **computational** viewpoint (CV), which is concerned with the functional decomposition of the system into a set of objects that interact at well-defined interfaces;
- the **engineering** viewpoint (NV), which is concerned with the infrastructure required to support distribution;

- the **technology** viewpoint (TV), which is concerned with the choice of technology used to implement the system and to connect it with its environment.

These viewpoints are of course mutually related, but no temporal order of their development is implied. They are (at least in theory) separately specified, and sufficiently independent to simplify reasoning about the complete system specification.

2.2 ODP Correspondences

ODP clearly states that a set of viewpoint specifications of an ODP system written in different viewpoint languages should not make mutually contradictory statements i.e., they should be mutually consistent.

The key to consistency is the idea of correspondences between different viewpoint specifications, i.e., a statement that some terms or structures in one specification correspond to other terms and structures in a second specification.

The requirement for consistency between viewpoint specifications implies that what is specified in one viewpoint specification about an entity needs to be consistent with what is said about the same entity in any other viewpoint specification. This includes the consistency of that entity's properties, structure and behavior.

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 (ISO/IEC 10746-3, ITU-T Rec. X.903, 1996):

- 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**.

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.

For the sake of completeness, the rest of the clauses of this section describe the correspondences between every pair of viewpoints, as defined in Part 3 of RM-ODP (ISO/IEC 10746-3, ITU-T Rec. X.903, 1996), the Enterprise Language (ISO/IEC 15414, ITU-T Rec. X.911, 2006), and in UML4ODP (ISO/IEC CD 19793, ITU-T Rec. X.906, 2005). Those familiar with ODP correspondences can skip the rest of this section.

2.2.1 EV and IV specification correspondences

In general, not all the elements of the enterprise specification of a system need to correspond to elements of its information specification. Where there is a correspondence between enterprise and information elements (e.g., between an enterprise object and the information object that stores the relevant information about it), the specifier has to provide:

EI-1 for each enterprise object in the enterprise specification, a list of those information objects (if any) that represent information or information processing concerning the entity represented by that enterprise object;

EI-2 for each role in each community in the enterprise specification, a list of those information object types (if any) that specify information or information processing of an enterprise object fulfilling that role;

EI-3 for each policy in the enterprise specification, a list of the invariant, static and dynamic schemata of information objects (if any) that correspond to the enterprise objects to which that policy applies (an information object is included if it corresponds to the enterprise community that is subject to that policy);

EI-4 for each action in the enterprise specification, the information objects (if any) subject to a dynamic schema constraining that action;

EI-5 for each relationship between enterprise objects, the invariant schema (if any) which constrains objects in that relationship;

EI-6 for each relationship between enterprise roles, the invariant schema (if any) which constrains objects fulfilling roles in that relationship.

2.2.2 EV and CV specification correspondences

Not all the elements of the enterprise specification of a system need to correspond to elements of its computational specification. In particular, not all states, behaviors and policies of an enterprise specification need to correspond to states and behaviors of a computational specification. There may exist transitional computational states within pieces of computational behavior which are abstracted as atomic transitions in the enterprise specification.

Where there is a correspondence between enterprise and computational elements, the specifier has to provide:

EC-1 for each enterprise object in the enterprise specification, the configuration of computational objects (if any) that realizes the required behavior;

EC-2 for each interaction in the enterprise specification, a list of those computational interfaces and operations or streams (if any) that correspond to the enterprise interaction, together with a statement of whether this correspondence applies to all occurrences of the interaction, or is qualified by a predicate;

EC-3 for each role affected by a policy in the enterprise specification, a list of the computational object types (if any) that exhibit choices in the computational behavior that are modified by the policy;

EC-4 for each interaction between roles in the enterprise specification, a list of computational binding object types (if any) that are constrained by the enterprise interaction;

EC-5 for each enterprise interaction type, a list of computational behavior types (if any) capable of representing (i.e. acting as a carrier for) the enterprise interaction type;

EC-6 if a process-based approach is taken, the specifier has to provide, for each step in the process, a list of participating computational objects which may fulfil one or more of actor roles, artefact roles, and resource roles.

2.2.3 EV and NV specification correspondences

Not all the elements of the enterprise specification of a system need to correspond to elements of its engineering specification. Where there is a correspondence between enterprise and engineering elements, the specifier has to provide:

EN-1 for each enterprise object in the enterprise specification, the set of those engineering nodes (if any) with their nuclei, capsules, and clusters, all of which support some or all of its behavior;

EN-2 for each interaction between roles in the enterprise specification, a list of engineering channel types and stubs, binders, protocol objects and interceptors (if any) that are constrained by the enterprise interaction;

EN-3 the engineering channel types and stubs, binders or protocol objects may be constrained by enterprise policies.

2.2.4 IV and CV specification correspondences

RM-ODP does not prescribe exact correspondences between elements of the information and computational specifications of a system. In particular, information objects do not need to correspond to computational objects. Likewise, not all states of an information specification need to correspond to states of a computational specification. There may exist transitional computational states within pieces of computational behavior that are abstracted as atomic transitions in the information specification. However,

IC-1 where an information object corresponds to a set of computational objects, static and invariant schemata of an information object may correspond to possible states of the computational objects; every change in state of an information object corresponds either to some set of interactions between computational objects or to an internal action of a computational object; and the invariant and dynamic schemata of the information object correspond to the behavior and environment contract of the computational objects.

2.2.5 IV and NV specification correspondences

The RM-ODP is silent about these correspondences.

2.2.6 CV and NV specification correspondences

ODP establishes some required correspondences between these two viewpoints.

CN-1 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.

CN-2 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.

CN-3 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.

CN-4 Each computational binding (either primitive bindings or compound bindings with associated binding objects) corresponds to either an engineering local binding or an engineering channel. This engineering local binding or channel corresponds only to that computational binding. If the computational binding supports operations, the engineering local binding or channel shall support the interchange of at least: computational signature names; computational operation names; computational termination names; and invocation and termination parameters (including computational interface identifiers and computational interface signatures).

CN-5 Except where transparencies that replicate objects are involved, each computational binding object control interface has a corresponding engineering interface and there exists a chain of engineering interactions linking that interface to any stubs, binders, protocol objects or interceptors to be controlled in support of the computational binding.

CN-6 Each computational interaction corresponds to some chain of engineering interactions, starting and ending with an interaction involving one or more of the basic engineering objects corresponding to the interacting computational objects.

CN-7 Each computational signal corresponds either to an interaction at an engineering local binding or to a chain of engineering interactions that provides the necessary consistent view of the computational interaction.

CN-8 In an entirely object-based computational language, data are represented as abstract data types (i.e., interfaces to computational objects). Computational interface parameters (including those for abstract data types) can be passed by reference, and then such parameters correspond to engineering interface references. Computational interface parameters can also be passed by migrating or replicating the object supporting the interface. In the case of migration such parameters correspond to cluster templates.

2.2.7 TV specification correspondences with the rest

There are neither required correspondences nor required correspondence statements between the enterprise, information or computational viewpoints, and the technology viewpoint. There may be cases, however, where part of one of these three viewpoint specifications has a direct relationship with a technology viewpoint specification, i.e, with a choice of technology. Examples include enterprise policies, invariant schemata, or computational environment contracts covering performance (e.g. response time), reliability, and security. In these cases, the appropriate correspondence statements between the related viewpoint specification elements should the defined.

RM-ODP defines the following correspondences between the engineering and the technology viewpoint specifications.

NT-1 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.

NT-2 Engineering interfaces correspond to technology interfaces.

2.3 Expressing Correspondences

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

Some of the proposals, e.g., (Boiten et al., 2000; Große-Rhode, 2004), 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 the correspondences, specially when viewpoint specifications are expressed as UML models, using different alternatives. One interesting possibility is the use of OCL to define relationships between the metamodel elements that represent the appropriate modeling concepts, as suggested by, e.g., (Dijkman, 2006). This approach works very well when the correspondences are defined between all the instances of certain modeling concepts, e.g., when every computational interface corresponds exactly to one engineering interface (correspondence CN-2). However, there are cases in which correspondences need to be established between particular objects of an specification. The problem is that it is not possible 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.

UML 2.0 *abstraction dependencies*, possibly constrained by OCL statements, are the natural mechanism provided by UML 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. 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 (Yahiaoui et al., 2005a; Yahiaoui et al., 2005b), 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. Therefore, Yahiaoui et al. define a new viewpoint, the *link* viewpoint, whose elements are "links" that establish binds between elements in different viewpoints. These links explicitly represent the ODP correspondences, and store the relevant information about the relationships between the views and the information related to each one (as attributes of the class that represents the link), thus guaranteeing traceability. A (change manager) tool has been developed for defining and enforcing these links, thus providing automated support for change management and propagation.

We do not think that such correspondences constitute another ODP viewpoint. ODP explicitly states that correspondences do not form part of any viewpoint. In addition, ODP defines the concept of viewpoint *on a system*, whilst correspondences are defined *between two viewpoints*. However, we do agree that correspondences should be represented by something more powerful than UML abstraction dependencies for the reasons stated above: correspondences may require to store more information than a single UML abstraction dependency can convey, and they may be required for other purposes—e.g., for enforcing and propagating changes from one view to another.

The fact that change propagations can be considered particular cases of model transformations suggests the use of QVT as the perfect solution to the problem of representing ODP correspondences. The use of relations was initially indicated by (Akehurst, 2004) for relating concepts from different viewpoint at the metalevel but not explored any further for relating instances, which is essential for establishing proper 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." (ISO/IEC 10746-3, ITU-T Rec. X.903, 1996)

3 QVT

3.1 QVT Relations

MOF QVT (Query/View/Transformation) (OMG, 2005) is the OMG's standard for specifying MOF model queries, views and transformations. It is expected to play a central role in the Model Driven Architecture (OMG, 2001). QVT defines three different (but closely related) languages for specifying transformations using declarative and imperative styles. Black-box implementations of operations can also be used to allow reuse of existing algorithms or domain specific libraries in certain model transformations.

QVT Relations is a language to write declarative specifications of the relationships between MOF models. The QVT Relations language supports object pattern matching, and implicitly creates trace classes and their instances to record what occurred during a transformation execution. Relations can assert that other relations also hold between particular model elements matched by their patterns.

QVT Relations allow for the following execution scenarios (OMG, 2005):

- Check-only transformations to verify that models are related in a specified way.
- Single direction and bi-directional transformations.
- The ability to establish relationships between pre-existing models, whether developed manually, or through some other tool or mechanism.
- Incremental updates (in any direction) when one related model is changed after an initial execution.
- The ability to create as well as delete objects and values, while also being able to specify which objects and values must not be modified.

3.2 QVT Transformations

In the relations language, a transformation between candidate models is specified as a set of relations that must hold for the transformation to be successful. A *candidate model* is any model that conforms to a *model type*, which is a specification of what kind of model elements any conforming model can have. An example is:

```
modeltype EL uses "odp.UML4ODP.EL_UMLProfile"
modeltype IL uses "odp.UML4ODP.IL_UMLProfile"
transformation EVtoIV (ev : EL, iv : IL) {
      top relation EVrole2IVobjectType {...}
      top relation EVobject2IVobject {...}
      ...
  }
```

Relations in a transformation declare constraints that must be satisfied by the elements of the candidate models, and specify a relationship that must hold between the elements of the candidate models. Top level relations are those that need to hold for a transformation to be successfully executed.

A relation is defined by two or more domains and a pair of when and where predicates. For instance, the following relation EVrole2IVobjectType establishes a relationship between roles in the EV specification and object types in the IV specification, whereby every enterprise role is related to one information object type with the same name (but not necessarily vice-versa, i.e., not every information object type should correspond to an enterprise role).

```
relation EVrole2IVobjectType {
   /* maps e-roles to i-objectTypes */
   domain ev er:Class {name=r}
   domain iv iot:Class {name=r}
   when {
      er.stereotypedBy("EV_Role")
   }
   where {
      er.stereotypedBy("EV_Role") and
      iot.stereotypedBy("IV_ObjectType")
   }
}
```

More precisely, relation EVrole2IVobjectType checks that for each role in the EV specification (i.e., a class stereotyped EV_Role) there is an object type with the same name in the IV specification (i.e., a class stereotyped IV_ObjectType).

A transformation can be invoked either to check two models for consistency or to modify one model to enforce consistency. In the first case, the transformation checks whether the relations hold in all directions, and report errors when they do not hold. In case of enforcement, one model acts as source and the other as target; the execution of the transformation proceeds by first checking whether the relations hold, and for relations for which the check fails, attempting to make the relations hold by creating, deleting or modifying only the target model, thus enforcing the relationship.

QVT transformations can also be used for propagating changes from one model to other. As mentioned in the QVT standard (OMG, 2005), "the effect of propagating a change from a source model to a target model is semantically equivalent to executing the entire transformation afresh in the direction of the target model. The semantics of object creation and deletion guarantee that only the required parts of the target model are affected by the change. Firstly, the semantics of check-before-enforce ensures that target model elements that satisfy the relations are not touched. Secondly, key-based object selection ensures that existing objects are updated where applicable. Thirdly, deletion semantics ensures that an object is deleted only when no other rule requires it to exist."

4 Modeling ODP Correspondences

We have seen how QVT transformations can be specified to define general relationships between elements of two ODP viewpoint specifications (e.g, between enterprise roles and information object types, or between enterprise objects and information objects). However, these kinds of correspondences are not very common in the specification of any ODP system. Usually, correspondences are defined between particular elements of the specification (e.g., between particular objects, types, templates, or actions).

For instance, suppose that we have an ODP specification of a Banking system, in which bank accounts are modeled in the computational viewpoint as objects that support a couple of interfaces for accessing their services. In the engineering viewpoint specification, we want each of these computational objects to correspond exactly to two basic engineering objects that support the same interfaces (plus possibly other interfaces only relevant to the engineering objects concerned). The specification of such part of the system at the object template level, and using the UML profiles defined in UML4ODP, is shown in Figure 1.

In order to represent such a correspondence, we could use a set of UML abstraction dependencies between the related elements. However, this could be done in a more precise and effective way using QVT.

At the object level, we need to define a relation that establishes a correspondence between a computational object which is an instance of an Account

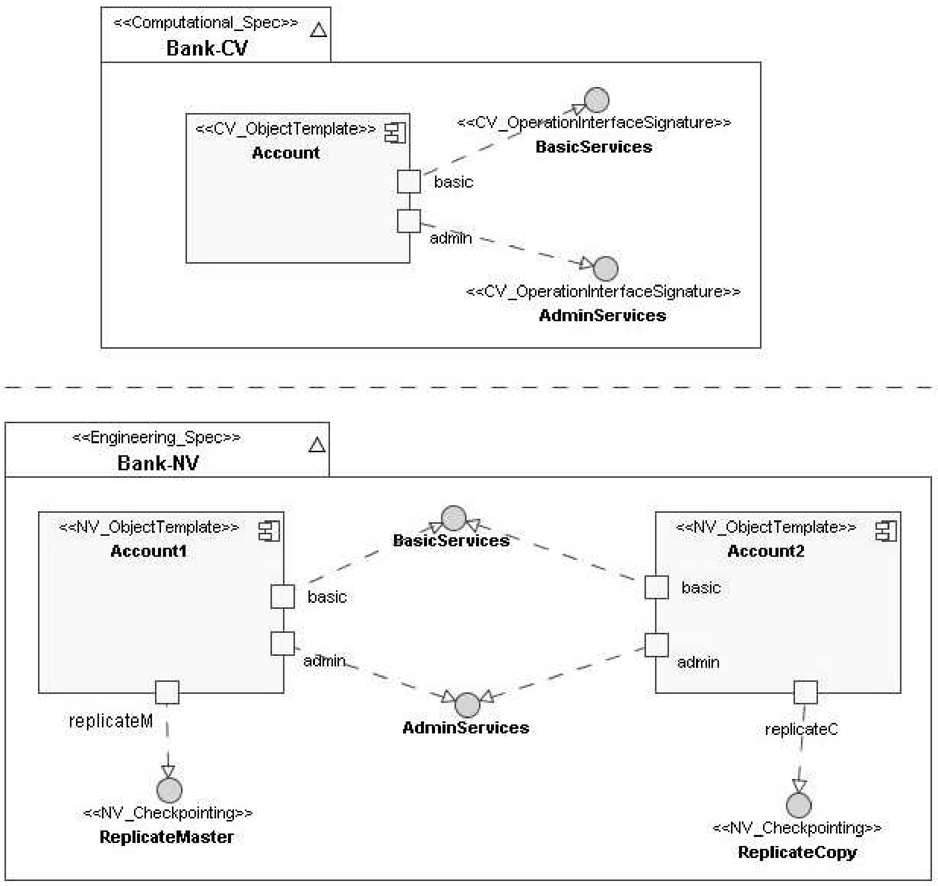

Figure 1: Bank Account comp. objects and interfaces should be related to the corresponding eng. objects and interfaces

object template, and two engineering objects that represent it in the engineering specification:

```
relation cv-account2twonv-accounts {
  domain cv a:InstanceSpecification
    {name=n, classifier = “Account”}
  domain nv a1:InstanceSpecification
    {name=n + ’1’, classifier = “Account1”}
  domain nv a2:InstanceSpecification
    {name=n + ’2’, classifier = “Account2”}
  when { a.stereotypedBy(”CV_Object”) }
  where {
    a.stereotypedBy(”CV_Object”) and
    a1.stereotypedBy(”NV_BEO”) and
    a2.stereotypedBy(”NV_BEO”) and
    DuplTemplates(a.classifier,a1.classifier,a2.classifier)
  }
}
```

We can see how it establishes that if there exists a UML InstanceSpecification stereotyped CV_Object, whose classifier is an Account, then there should be two UML InstanceSpecifications stereotyped NV_BEO, whose classifiers are Account1 and Account2, respectively. In addition, a relation called DuplTemplates should also hold between the classifiers of all these instance specifications. Such a QVT relation is precisely the one that establishes the correspondence between the appropriate computational object templates (Fig. 1):

```
relation DuplTemplates{
  domain cv a:Component {name=n}
  domain nv a1:Component {name=n + ’1’}
  domain nv a2:Component {name=n + ’2’}
  when { a.stereotypedBy(”CV_ObjectTemplate”) }
  where {
    a.stereotypedBy(”CV_ObjectTemplate”) and
    a1.stereotypedBy(”NV_ObjectTemplate”) and
    a2.stereotypedBy(”NV_ObjectTemplate”) and
    sameODPInterfaces(a,a1) and
    sameODPInterfaces(a,a2)
  }
}
```

This relation establishes that a given computational object template should be related to two engineering object templates (whose names should be the same, but suffixed with '1' and '2'), and that the ODP interfaces of the computational object template should be supported by the corresponding interfaces of the engineering object templates—as stated by the ODP required correspondence **CN-3**. This required correspondence is expressed using the sameODPInterfaces relation, that checks that every interface defined for a computational object template is supported by an interface of a given engineering object template. In the UML4ODP context, both computational and engineering object templates are modeled using UML components, and both computational and engineering interfaces are represented by UML ports. Thus, the QVT relation checks that every port of the UML component representing the computational object template has an associated port with the same name in the given UML component representing the basic engineering object template, and that the set of provided and required interfaces of each port are the same in the two specifications.

```
relation sameODPInterfaces {
  domain cv cot:Component {}
  domain nv eot:Component {}
  when {
    cot.stereotypedBy("CV_ObjectTemplate") and
    eot.stereotypedBy("NV_ObjectTemplate")
  }
  where {
    eot.ownedPort.name->includes(cot.ownedPort.name)
    and cot.ownedPort->forAll(p | p.required =
        eot.ownedPort->select(name=p.name).required)
    and cot.ownedPort->forAll(p | p.provided =
        eot.ownedPort->select(name=p.name).provided)
  }
}
```

This last relation can be reused as-is in other QVT relations to enforce the required correspondence, **CN-3**, in other ODP correspondence statements.

5 Issues for Discussion

Once we have briefly seen how QVT could be used to represent both ODP correspondence statements and ODP required correspondences, let us discuss in this section some issues that may require further investigation.

5.1 Bi-directionality and cardinality of correspondences

The RM-ODP is silent about the possible bi-directionality of the ODP correspondences. However, we believe such correspondences must be bidirectional so it is possible to navigate from any of the two views to the other. The idea is to be able to trace elements, i.e., given an element of a viewpoint, find all the elements in the rest of the viewpoints which are related to it (objects, policies, rules, actions, etc.).

In addition, RM-ODP seems to define correspondences just between pairs of viewpoints. However, sometimes correspondences between one and more viewpoints might be required, i.e., between one element in one viewpoint and several elements in other viewpoints. Defining this kind of 1-M correspondences is possible with QVT relationships, although something not defined in RM-ODP.

5.2 Transitivity of correspondences

The QVT relations presented here can be used for change propagation. This occurs when a change happens in one of the viewpoint specifications, and we want to propagate the change to all related elements in the rest of the viewpoint specifications. In this case we can consider QVT relations as model transformations, enforcing the relationships on the target models as mentioned earlier. However, this may raise some redundancy or duplication issues due to transitivity of the relations.

Suppose elements α, β and γ in viewpoints A, B and C respectively, related as follows: α is related with β and γ, and β is related with γ. How to deal with the potential redundancy that may happen when a change in element α is propagated to γ both directly from α to γ, and indirectly through β? There are cases where this does not imply any problem, as it happens when the relations just check that the elements have the same name, and we change the name of α. However, what happens when the relations add something to the elements' structure or behavior? E.g., suppose they add a suffix to the name of the element? Will we end up with a duplicated suffix in the name of γ?

Please notice how this is an example that could justify the need for establishing N-M correspondences between viewpoints.

5.3 Full consistency of specifications

In order to check the consistency of the specifications, we can use the ODP correspondences if we consider them as model transformations, as mentioned in the RM-ODP standards. However, complete consistency between viewpoint specifications cannot be guaranteed by ODP correspondences only. Analysis of consistency depends on the application of specific consistency techniques, most of which are based on checks for particular kinds of inconsistency, and thus cannot prove complete consistency.

This latter issue has been addressed by several people, from different perspectives. The interested reader can consult, e.g., the works by Derrick, Bowman et al. (Boiten et al., 2000), the interesting book (Große-Rhode, 2004), and also the recent and complete work done by Remco Dijkman in his PhD thesis (Dijkman, 2006). How to combine the use of model-driven techniques and QVT in those contexts is something we would like to explore further as part of future research.

6 Conclusions

In this paper we have sketched how QVT relations can be used to represent ODP correspondences in the context of the UML4ODP project, in an initial attempt to show that this approach is feasible. QVT relations provide more powerful mechanisms than those provided by plain OCL or UML abstraction dependencies for relating elements in different ODP viewpoints, can be modularly and independently specified, be reused to build more powerful QVT transformations, and serve both for checking the correspondences and for enforcing them.

There are still several issues open for investigation. Apart from the questions mentioned above, it is not clear whether this method is better or not than the other ones discussed here, e.g., the one proposed by Remco Dijkman (Dijkman, 2006), or by Yahiaoui et al. (Yahiaoui et al., 2005a; Yahiaoui et al., 2005b). Furthermore, apart from specifying the correspondences, can the QVT relations provide any other advantages? Can they be used, for instance, to reason about the system specifications and their consistency? And if so, how this can be achieved? Which is the underlying logic in which the reasoning can be done? Apart from consistency, what other properties can be proved from the QVT specifications of the correspondences? These are interesting questions, some of them we plan to address in a near future.

Acknowledgements The authors would like to thank the anonymous referees for their constructive comments and suggestions, that greatly helped us improve the contents and readability of the paper. This work has been partially supported by Spanish Research Project TIN2005-09405-C02-01.

REFERENCES

Akehurst, D. H. (2004). Proposal for a model driven approach to creating a tool to support the RM-ODP. In *Proc. of the 1st International Workshop on ODP in the Enterprise Computing (WODPEC)*, pages 65–68, Monterey, California.

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

Bowman, H., Derrick, J., Linington, P., and Steen, M. W. (1995). FDTs for ODP. *Computer Standards & Interfaces*, 17:457–479.

Dijkman, R. (2006). *Consistency in Multi-Viewpoint Architectural Design*. PhD thesis, University of Twente.

Große-Rhode, M. (2004). *Semantic Integration of Heterogeneous Software Specifications*. Springer-Verlag, Berlin.

IEEE Std. 1471 (2000). *Recommened Practice for Architectural Description of Software-Intensive Systems*. IEEE Standards Association.

ISO/IEC 10746-1 to 10746-4, ITU-T X.901 to X.904 (1997). *RM-ODP. Reference Model for Open Distributed Processing*. ISO & ITU-T.

ISO/IEC 10746-3, ITU-T Rec. X.903 (1996). *Information technology – Open distributed processing – Reference model: Architecture*. ISO & ITU-T.

ISO/IEC 10746-4, ITU-T Rec. X.904 (1998). *Information technology – Open distributed processing – Reference model: Architectural Semantics*. ISO & ITU-T.

ISO/IEC 15414, ITU-T Rec. X.911 (2006). *Information technology – Open distributed processing – Reference model – Enterprise language*. ISO & ITU-T.

ISO/IEC CD 19793, ITU-T Rec. X.906 (2005). *Information technology – Open distributed processing – Use of UML for ODP system specifications*. ISO & ITU-T.

Linington, P. (1995). RM-ODP: The architecture. In Milosevic, K. and Armstrong, L., editors, *Open Distributed Processing II*, pages 15–33. Chapman & Hall.

OMG (2001). *Model Driven Architecture. A Technical Perspective*. Object Management Group. OMG doc. ab/2001-01-01.

OMG (2005). *MOF QVT Final Adopted Specification*. Object Management Group. OMG doc. ptc/05-11-01.

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

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