Realizing Correspondences in Multi-Viewpoint Specifications

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

Juan Ignacio Jaén, Antonio Vallecillo
Dept. Lenguajes y Ciencias de la Computación,
Universidad de Málaga, Spain
{jijaen, 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 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. This paper proposes a generic model-driven approach to the specification and realization of correspondences between viewpoints. In particular, we show how correspondences can be modeled both extensionally and intensionally, and propose the use of model transformations to connect these two approaches. As a proof-of-concept, we show how our proposal can be implemented in the context of the RM-ODP and UML4ODP, and present a tool to support the realization of correspondences between ODP views. This proposal can be extended to any other Architectural Framework that uses models to represent their views.

1. Introduction

Large-scale heterogeneous distributed systems 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 [16]. 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 [23], Viewpoints [12], OpenViews [6], Dijkman's framework [7], or the growing plethora of Architectural Frameworks (AFs): the Zachman framework [42], ArchiMate [24], 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 is an architectural practice jointly defined by ISO/IEC and ITU-T that provides five generic and complementary viewpoints on the system and its environment [18]. 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 for 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. 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.

Most viewpoint modeling approaches to system specification (including the IEEE 1471 standard itself and the majority of the existing AFs) do not consider correspondences between viewpoints, or assume they are trivially based on name equality between correspondent elements and thus they 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., [8, 9, 10, 12, 14, 37]) 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.

There are basically two approaches to model correspondences between the views of a system: extensional and intensional. *Extensional* approaches model correspondences between the particular elements of the views, similarly to what is done for 2D representation of 3D figures. However, in large systems the number of correspondences jeopardizes their proper design, management and maintenance: the system designer cannot deal with (or even properly define and visualize) thousands of correspondences. *Intensional* approaches define correspondences as relations between types of model elements, i.e., between viewpoint elements and not between view elements. However, this approach may hinder the understandability and operability of the specifications produced: for typical users of the specification, correspondences are easier to use, visualize and understand when they are drawn as relationships between individual elements in the views, instead of being expressed as formulae.

This paper examines a generic model-driven proposal to explicitly specify correspondences between viewpoints, reviews the possible approaches, and proposes the use of model transformations to connect the intensional and extensional specification of correspondences. In addition, we discuss two different and complementary ways to model extensional correspondences between views, in case they are represented as UML models: using classes and using dependencies. Each one is more apt for a particular task.

Traditionally, AFs have not formally dealt with correspondences. The RM-ODP is one of the few architectural frameworks that consider their explicit definition, and for which modeling tool support currently exists [30]. Thus, the work presented in this paper has been carried out in the context of the RM-ODP and UML4ODP. To validate our proposal and to serve as a proof-of-concept of our ideas, we show here how the existing modeling tool has been extended to support the specification of correspondences. It is also important to note that this proposal is not restricted to ODP: it can be easily applied in any other AF that uses models to represent its views.

After this introduction, Section 2 presents the concepts related to viewpoint correspondences and the way they are defined in RM-ODP. Then, Section 3 proposes the specification of viewpoints, views and correspondences using a metamodeling approach. Section 4 presents some relevant related work in this area. The details of our approach are introduced in Section 5. Finally, Section 6 compiles our conclusions and outlines some future work.

2. Viewpoint Correspondences

The most general approach to viewpoint consistency is based on the definition of correspondences between viewpoint elements. Historically, correspondences have not been handled by multi-viewpoint architectural frameworks. A well-known exception is the RM-ODP standard, that provides a precise conceptual framework and includes correspondences as first class citizens. The forthcoming standard IEEE and ISO/IEC 42010 [21], which is expected to update the current IEEE Std. 1471, will also include the notion correspondence between viewpoints, in a similar fashion to that of 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. Correspondences do not form part of any one of the viewpoints, but provide statements that relate the various viewpoint specifications—expressing their semantic relationships [25]. Hence, we could initially say that a proper system specification consists of a set of viewpoint specifications, together with a set of correspondences between them.

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. Examples of *required correspondences* are defined in [18] between pairs of viewpoints. For example, the next two correspondences are defined for all the existing elements in the computational and engineering viewpoints:

C1. 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.

C2. 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.

These correspondences are stated within the specific context of ODP. For the sake of simplicity we will not describe in detail the ODP concepts involved in these correspondences. The interested reader can consult Part 3 of RM-ODP [18], the Enterprise Language [20] and UML4ODP [19] for the complete set of correspondences between pairs of viewpoints defined by the RM-ODP. In any case, we want to emphasize the fact that these required correspondences establish certain constraints on *every* instance

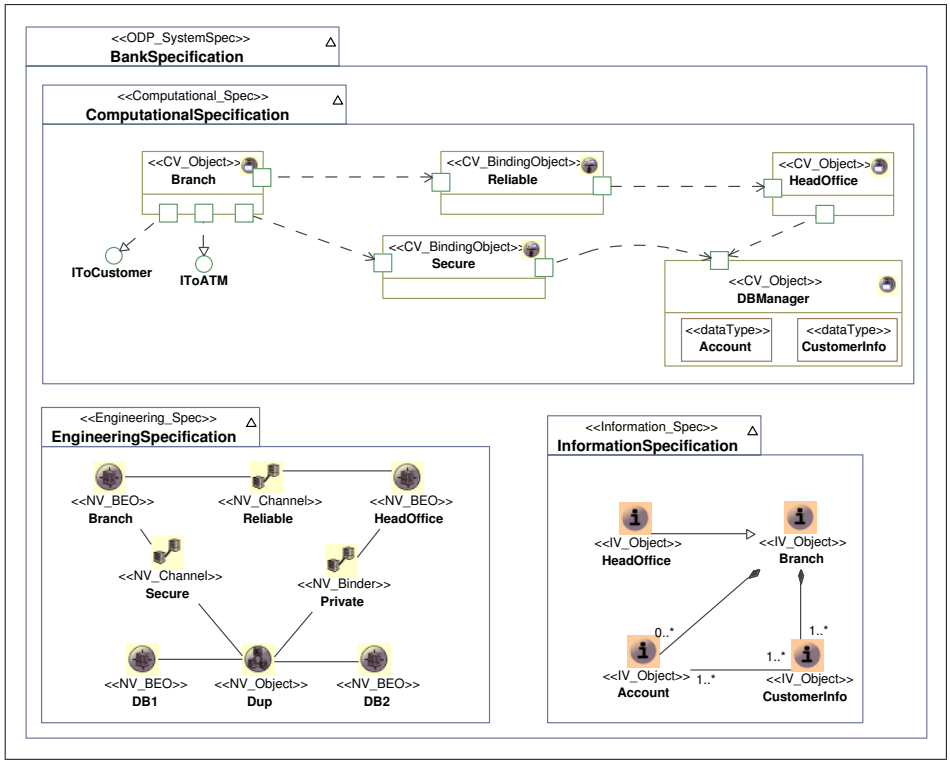

Figure 1. Example of a Bank specification

of particular types of elements, as mentioned above, e.g., computational objects, engineering interfaces, etc.

In general, correspondences are key elements of any multi-viewpoint AF. Consequently, we need to count on approaches to specify and realize them, supported by the appropriate tools. This is precisely the main aim of this paper. Furthermore, correspondences can also be used for other purposes, e.g., change management in multi-view systems [4, 11, 15]. 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 occurs in any of them, i.e., propagating the changes to maintain consistency. This is something our approach is also expected to provide with the appropriate connection to model synchronization tools such as reSynch [33].

3. Viewpoint, Views and Correspondences

Based on our previous work [32], in this section we formulate the specification of viewpoints, views and correspondences from a metamodeling approach, in order to be able to tackle some of the problems related to their specification. 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 [5, 34]. 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.

A running example. In order to illustrate our proposal we will use here a simple example of a multi-view specification in the context of the RM-ODP. It models a banking application, which manages accounts owned by customers. Branches store this information. A head office is a kind of branch with special functionality (not described here for simplicity). The basic information of the system is described in the Information Viewpoint specification. The Computational Viewpoint focuses on the functionality of

the system, whereas the Engineering Viewpoint deals with how this functionality is distributed. A first step consists of building all these views. Fig. 1 shows three models, one for each of these three views, using the UML Profiles defined by the UML4ODP standard [19].

The way to specify correspondences is by using models, too, which conform to the appropriate metamodels. Such correspondence metamodels can be defined using for instance the approach defined in UML4ODP shown in Fig. 2, OCL constraints [7, 26], or Model Transformation languages to define viewpoint correspondences as model transformations (e.g., using QVT [28, 31]). See Section 4 for a more detailed description of these approaches.

From a modeling perspective we can see that initial approaches (including, e.g., the Zachman framework, IEEE Std. 1471 and most Enterprise AFs: TOGAF, DoDAF, FEAF, etc.) define that a multi-viewpoint specification consists of a set of views of the system, each one expressed as a model that conforms to the metamodel of its associated viewpoint.

Although the relationships (i.e., correspondences) between the views are sometimes mentioned, these approaches define neither precise concepts and mechanisms for specifying correspondences, nor notations for modeling them. In this sense, correspondences are *implicitly* defined in these approaches. Other proposals, such as those mentioned in Section 4, or the new IEEE and ISO/IEC 42010 propose the explicit specification of correspondences between viewpoints. However, this approach does not permit the specification of the required correspondences, which describe the well-formed rules that the set of correspondences between elements of views should obey. Therefore, this justifies the following definition of multi-viewpoint system specification [32]:

Definition 1 *A* Multi-Viewpoint 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 $\mathcal{C}$*[1]*. Rules are expressed as constraints on the correspondence elements, using any constraint language (e.g., OCL).*

The problem now is to endow the system specifier with notations and tools to express the views, the correspondences and the well-formed rules on them.

[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.

4. Previous Works on Correspondences

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 AFs), or implicitly defined using the names of the related elements (e.g., [8, 9, 10, 12, 14, 37]). The problem is that without explicitly representing correspondences we cannot reason about them, nor properly tackle the integration and consistency issues mentioned above.

Another interesting issue is that the majority of these approaches assume that we can build an underlying metamodel containing all the views, in order to deal with the problem of inconsistency among viewpoints. However, this is not normally true. From a theoretical perspective, the use of a common global metamodel greatly helps maintaining the coherence and conceptual integration among viewpoint elements. However, the definition of such an underlying metamodel presents some problems. Firstly, should the metamodel consist of the intersection or of the union of all viewpoints elements? Some proposals (e.g., ArchiMate [24]) use the first approach (i.e., the intersection), while others, e.g., Dijkman [7] or Grosse-Rhode [14], use the second. Both approaches have serious problems with the extensibility and expressiveness of the basic elements of the global metamodel (not to mention complexity of the second approach—think for instance of the UML 2 metamodel). Secondly, defining a common metamodel can be feasible if the granularity and level of abstraction of the viewpoints is similar and not arbitrarily different, something which cannot be guaranteed in complex AFs such as Zachmann, or those that allow nested levels of abstraction, such as RM-ODP. Finally, the viewpoints may have very different formal semantics, which greatly complicates the definition of the common underlying metamodel. Basically, we conclude that a common metamodel is feasible if the viewpoints are (semantically) tightly coupled, but rather artificial if they are loosely coupled or describe the system at very different levels of abstraction or of granularity.

Several authors have recognized the independence of the individual viewpoints, and proposed different approaches to explicitly express correspondences between viewpoint elements for addressing viewpoint consistency checking. Some of the proposals, e.g. [6, 12, 14, 24, 37], 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. We will distinguish be-

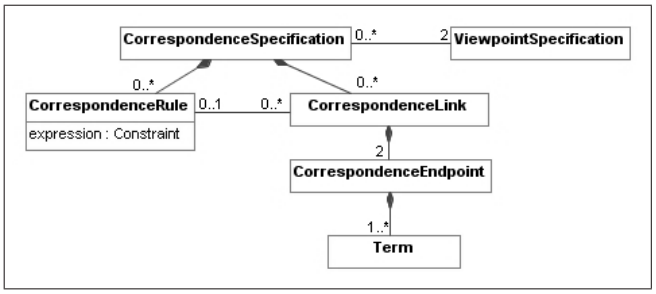

Figure 2. Correspondence metamodel [19]

tween extensional and intensional approaches.

Extensional approaches define correspondences between individual view elements, and are usually the natural way in which correspondences are drawn. For instance, if views are expressed as UML models, the UML 2 language defines *abstraction dependencies*, possibly constrained by OCL statements, as the natural mechanism to model a relationship that relates two elements or sets of elements that represent the same concept at different levels of abstraction or from different viewpoints [27]. This approach works well for relating individual elements in two views. However, it does not allow simultaneously relating *sets* of elements in each viewpoint, something required in some situations.

An alternative approach to represent correspondences has been defined by ISO/IEC and ITU-T in the context of the UML4ODP standardization project [19]. The UML4ODP correspondence metamodel is shown in Fig. 2.

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.

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 sets of related elements (to achieve, e.g., consistency management and synchronization enforcement).

This approach is not free from drawbacks, however. For example, 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. In addition, the fact that it works at model level hampers the easy and rapid definition of correspondences between particular types of objects, e.g., between *all* computational and engineering objects. Finally, this approach does not scale well. As soon as the number of elements in a system specification is high (which is normally the case), the number of correspondences that have to be specified and maintained grows exponentially. And this makes the complete system specifications hard to specify, understand, and maintain.

Intensional approaches are usually defined as relations between types of model elements, i.e., between viewpoint (or metamodel) elements and not between individual view elements. In other words, if we consider that a view is a model and a viewpoint is the metamodel for that view, intensional approaches relate elements at metamodel level.

This approach has been proposed by several authors for relating concepts from different viewpoint at the metalevel (as initially suggested by Akehurst [2, 3] using relations defined in OCL). Dijkman [7] also used relations and consistency rules in his framework for preserving consistency among viewpoints.

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, at least at metamodel level. As explained in Section 3, in a previous work [31] we explored the use of QVT for defining viewpoint correspondences as model transformations. The main benefit of this approach is that it allows checking pairwise consistency between related viewpoints using standard mechanisms and tools.As drawbacks, QVT is not free of semantic problems [36].

Intensional approaches work very well, for instance, when every object of a certain type in a given view is related to another object in another view (i.e., when relations can be defined at viewpoint, or metamodel, level). However, there are cases in which correspondences need to be established between particular objects of a specification (as it happens when the user defines the specific objects in one view that should be replicated in another view, see e.g., required correspondence **C2** above). The problem is that at the metalevel it is not that simple and elegant to determine which particular objects should be related.

Another problem (probably the most critical one) with the intensional specification of correspondences is related to the learnability, usability and maintainability of the spec-

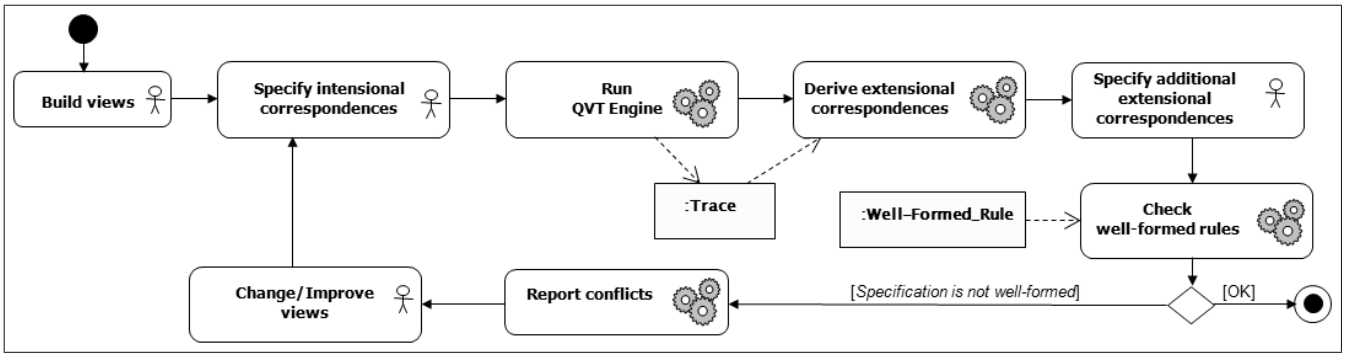

Figure 3. Overview of the multi-viewpoint specification process

ifications produced. For typical users of the specification, it is easier to use and understand when correspondences are drawn between individual elements in the views, instead of by a set of model transformation rules. For instance, in the ODP context, given an object in a computational view, we would like to know which are the enterprise policies in the enterprise view that constraint its behavior. Similarly, given an enterprise policy we can be interested in identifying the elements in the rest of the views that are affected by that policy. Therefore, it is important that correspondences can be established between specific model elements, too.

Furthermore, there are many situations in which correspondences can not be specified as model transformations because they are not functions—rather, they are just data mappings between related elements but without any transformation or change propagation mechanism defined between them. In this latter case, model weaving techniques [1] can be more appropriate than model transformations for expressing correspondences.

In summary, neither of the two individual approaches (intensional and extensional) is free from problems. We need to count on both for expressing correspondences. And we cannot forget about the need to model the well-formed rules on the set of correspondences [32].

5. Our approach

Assuming that views are expressed as UML models, our solution uses QVT Relations for intensionally specifying correspondences; UML dependencies for modeling them extensionally; and OCL for specifying the well-formed rules on the set of extensional correspondences. Fig. 3 summarizes the specification and transformation process from intensional to extensional specifications. Some of the steps (those marked with gears) are fully automated. Activities marked with a little stick figure represent actions that count with tool support, but need to be performed by human actors (the designers or software architects specifying the system). All these steps are discussed next.

Note that this approach is conceived for those frameworks whose viewpoint languages and relationships among their respective elements are described in terms of metamodels. Although most current architectural frameworks are not described in this way (many can be considered just as *conceptual* frameworks), this is not, however, a too strong assumption because of the increasing interest today in model-driven approaches. Thus, different organizations and groups are trying to "metamodel" their proposals, or at least to provide some kind of abstract syntax (e.g., the MODAF Meta-Model, M3 [38]). Some of them also provide concrete syntaxes, e.g., ISO's UML Profile for ODP [19], UML4ODP, or the OMG's UML Profile for DoDAF and MODAF, UPDM [29].

Modeling correspondences intensionally with QVT. MOF QVT (Query/View/Transformation) [28] is the OMG's standard for specifying MOF model queries, views and transformations. 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. In [31] we showed how QVT can be used for defining viewpoint correspondences in the context of the RM-ODP, using QVT relations for specifying correspondences between elements at metamodel level (and hence relating all instances of particular viewpoint concepts). In this way, the *required* correspondences that RM-ODP (or any other architectural framework) defines can be easily expressed with this approach, e.g., that every engineering object must be related to one computational object, etc.

We also showed [31] that QVT can also be used for specifying correspondences between specific view elements (e.g., between particular objects, types, templates, or actions). For instance, the following QVT relation specifies

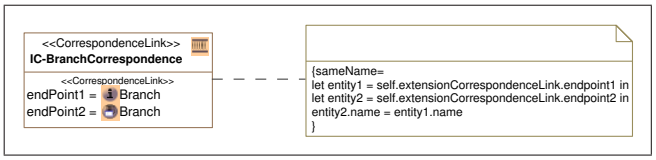

Figure 4. Example of a UML4ODP correspondence specification

a correspondence which establishes that Branch instances in the Computational view should be related to Branch objects in the Information view.

```
relation IC-BranchCorrespondence {
  domain iv iBranch:Class {name="Branch"}
  domain cv cBranch:Component {name="Branch"}
  when {
     iBranch.stereotypedBy("IV_Object") and
     cBranch.stereotypedBy("CV_Object")
  }
}
```

Although we showed that this approach to specify correspondence is feasible, we also learnt that it cannot be of practical use in general. Specifying correspondences between view elements in particular systems is a cumbersome task, not to mention the intrinsic difficulty involved in the tasks of understanding, visualizing and maintaining the QVT relations (compare this QVT relation with the equivalent correspondence described using the UML4ODP profile, which is shown in Fig. 4).

Extensional specification of correspondences. As we mentioned above, probably the most expressive way to specify correspondences in an extensional manner is by using an approach similar to that of the UML4ODP, in which correspondence links are represented by classes that contain all the required information about the correspondence.

An example of a correspondence specification using the UML4ODP notation is depicted in Fig. 4. It shows the same correspondence that was used to illustrate the use of QVT for modelling correspondences between view elements. It establishes that Branch instances in the Computational view should be related to Branch objects in the Information view. In particular, they should have the same name in our example.

The problem, again, is that defining correspondences extensionally in this way is too complex and cumbersome because the number of correspondences can grow exponentially as the number of model elements increases. Moreover, UML4ODP correspondences provide all the expressiveness required, but are hard to manage in a visual way.

In our proposal we make use of UML 2 *abstraction dependencies* [27] to provide an alternative representation of correspondence links. We have defined a UML profile for modeling correspondences, that uses UML dependencies (stereotyped ≪CorrespondenceLink≫) to represent them. UML dependencies are easier to draw and visualize, more intuitive than correspondence classes (as in UML4ODP) or OCL expressions, and provide the functionality required to define correspondence between views in most cases. Fig. 5 shows how the correspondences between the different views of the banking application example are specified with our approach. Compare this diagram with the equivalent representation of the correspondences using 11 UML4ODP ≪CorrespondenceLink≫ classes, something harder to understand and to manage.

In addition, in the case of the RM-ODP this notation can be easily (and automatically) transformed into the more expressive UML4ODP correspondences, therefore providing UML4ODP users with a more usable notation for expressing their correspondences. To connect this new profile with the UML4ODP profile, we have developed a set of model transformations. These model transformations have been defined in both directions, so that users can specify their correspondences using UML ≪CorrespondenceLink≫ dependencies, which are transformed into UML4ODP correspondence links, and vice-versa. One of the benefits of this approach is that it can work for any architectural framework whose models can be expressed in UML, or using UML profiles. Notice that correspondences can be defined between sets of elements in each related viewpoint. Although in theory UML 2 dependencies allow multiple cardinality in both ends [27, 7.3.12], this is often not supported by most UML modeling tools.

Finally, we already mentioned that one potential use of correspondences is to propagate changes in evolution scenarios. To address this additional issue, we can make use of the fact that dependencies are directed relations, and that the client of the dependency "depends" on the supplier. This establishes a semantic relationship between them that can indicate which one should change if the correspondence enforces a modification for one of them. UML dependencies can be endowed with a constraint that establishes the relationship that must hold between the elements related by the correspondence (i.e., the correspondence rule). The fact that most of these relationships are common and reusable has allowed us to characterize them and build a tool that can help synchronize general UML models using this kind of correspondences [33].

Transforming QVT relations into extensional correspondences. We also need to provide a mapping between the intensional and extensional specifications of correspondences, so that users can derive the initial set of extensional

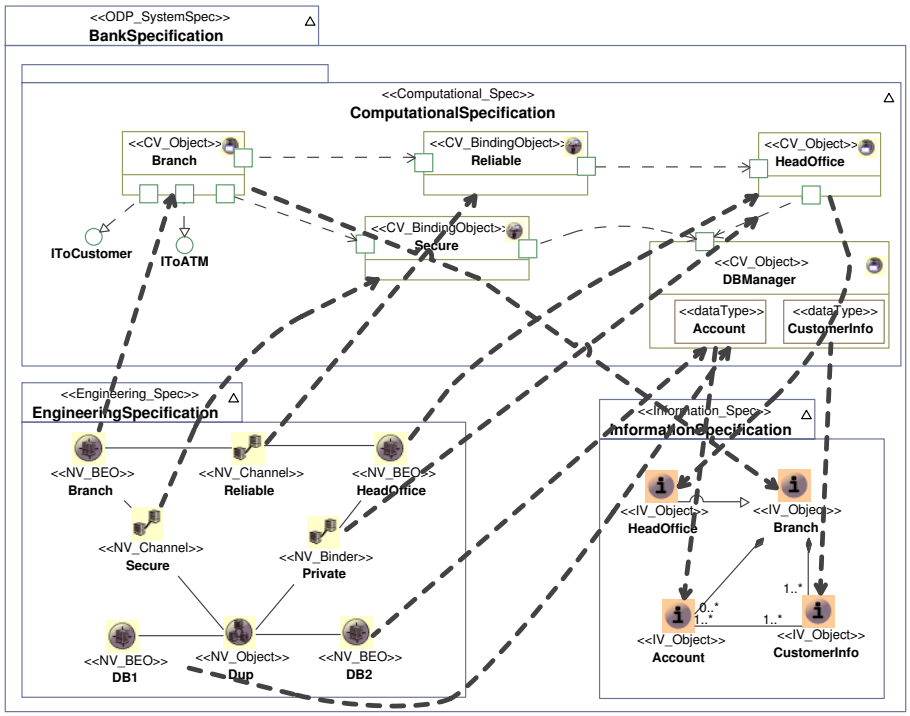

Figure 5. The Bank Specification With Correspondences.

correspondences from the intentional ones (which normally capture just the set of required correspondences for a particular architectural framework).

The way to relate the intensional and extensional specifications is based on the use of QVT Trace classes. In the context of the Relations Language, associated to each QVT relation there exists a *Trace class* in the QVT Core language [28]. The trace class contains a property corresponding to each object node in the pattern of each domain of the relation. Instances of these classes (called *trace instances*) are created during the execution of a transformation so that relationships between models that are created by the execution can be stored. In fact, a trace instance represents the linkage between models established by a transformation execution. QVT mentions that these instances may be used to aid in propagating incremental updates to a source model into a target model without re-executing the entire transformation. But, in our jargon, they also provide the extensional specifications of the set of correspondences defined by an intensional QVT specification.

QVT mentions that traces between model elements involved in a transformation are created implicitly in the Relations language, and that it is in the Core language that all trace classes are explicitly defined as MOF models, and trace instance creation and deletion is defined in the same way as the creation and deletion of any other object. However, the traces associated to a Relations transformation can also be made persistent, as some of the QVT implementations allow. With them, a model transformation that transforms QVT trace instances into, e.g., UML4ODP correspondence links can be easily defined. In this way, intensional specifications of correspondences can be transformed into their corresponding extensional specifications.

We are currently using Medini QVT [17], one of the few implementations of QVT Relations that supports the creation of persistent traces. Our tool invokes the Medini QVT engine to generate the traces, and then converts them into extensional correspondence links.

Expressing well-formed rules. 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 [18] (examples of these are **C1** and **C2** previously described in Section 2), and a set of user-defined statements for the ODP system being modeled.

The OCL language seems to be a very suitable alternative for expressing well-formed rules, specially if viewpoint

metamodels are MOF-conformant. OCL 2.0 is now aligned with MOF, and it counts with tool support. Examples of well-formed rules defined using OCL in the case of ODP and UML4ODP can be found in [32].

Putting All Pieces Together with Tool Support. An important issue for the adoption of any architectural approach is the availability of tools to support the development, storage, presentation, analysis, improvement and evolution of architecture representations. As with architecture methodologies, architecture tools to support the architectural development process are still emerging. As a response to these needs, we have developed of an integrated tool to provide support to our proposal in the context of the UML4ODP initiative, which is publicly available [30]. This tool has been recently accepted by MagicDraw and is one of the plugins currently available to download from the MagicDraw site, too.

This tool provides several interesting functionality to UML4ODP modelers. Firstly, it allows users to draw their models and validate their ODP specifications, i.e., check that they conform to the RM-ODP standard, and that the individual views are consistently specified.

Secondly, the tool provides full support to the correspondence specification process shown in Fig. 3. Thus, users can transform the intensional specifications of correspondences to their corresponding extensional specifications (expressed as UML4ODP correspondence links), using the Medini QVT as mentioned above for generating the traces, and then we transform them into UML4ODP correspondence links. These are in turn transformed into the corresponding UML dependencies stereotyped ≪CorrespondenceLink≫. Thus, users can specify, visualize and manage the correspondences between the view elements using any of the approaches discussed here, while the tool keeps them in synch.

6. Conclusion and Future Work

This paper addresses the problem of providing precise specifications of correspondences between viewpoints. We have discussed how correspondences can be specified in several ways, and the pros and cons of each approach. Our proposal tries to combine the different approaches, using model transformations to obtain ones from the others.

We have presented our proposal as a generic model-driven proposal to describe multi-viewpoint specifications, including the explicit specification and transformation of correspondences between viewpoints. Although the study case and the tool presented here have been implemented in the context of the RM-ODP, we believe that our approach can be easily adopted by the rest of the multi-viewpoint AFs, especially in those that use models to represent their views: TOGAF, DoDAF, FEAF, etc. Furthermore, the revision of IEEE Std. 1471, currently undertaken by ISO and IEEE and that will produce the new International Standard ISO/IEC 42010, already contemplates correspondences, and the work presented here is aligned with it.

As part of our future plans, we expect to extend this proposal and its supporting tool in the context of model synchronization and configuration management to realize, e.g., version control and change traceability and visibility of viewpoint and correspondence specifications. An interesting experiment has been conducted in the context of UWE models already, which have showed that propagating changes using UML dependencies annotated with different kinds of synchronization constraints is possible [33]. We plan to extend our dependencies with this kind of information, being able to connect our tool with the reSynch synchronizer and therefore provide full support for viewpoint synchronization of ODP views.

Another issue that needs to be further studied is the reverse transformation process, from extensional to intensional specifications of correspondences. Usually intensional correspondences are defined as relations between *types* of model elements, and this is done only once for every architectural framework. However, there are situations in which the user wants to start the process by specifying correspondences in an extensional way. The question here is how to transform these extensional specifications into intensional ones. In this context, we are working on the use of the so-called *model transformation by example* approach [39], so that intensional specifications can be built from extensional ones, either from scratch [40, 41, 13, 22] or using an approach that allows a refinement of the intensional specifications when a change in the extensional set of correspondences happens [35].

Acknowledgements. The authors would like to thank the anonymous reviewers for their insightful and constructive comments. This work has been partially supported by Spanish Research Projects TIN2008-03107, TIN2008-00889-E and P07-TIC-03184.

References

[1] A. Abouzahra, J. Bézivin, M. D. D. Fabro, and F. Jouault. A Practical Approach to Bridging Domain Specific Languages with UML profiles. In *Proc. of the BP-MDSD Workshop at OOPSLA'05*, 2005.

[2] 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.

[3] 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.

[4] H. Basson. A change propagation model and platform for multi-database applications. In *Proc. of ICSM'01*, 2001.

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

[6] 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.

[7] R. Dijkman. *Consistency in Multi-Viewpoint Architectural Design*. PhD thesis, University of Twente, 2006.

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

[9] A. Egyed. Instant consistency checking for the UML. In *Proc. of ICSE '06*, pages 381–390, 2006.

[10] A. Egyed. Fixing inconsistencies in UML design models. In *Proc. of ICSE'07*, pages 292–301, 2007.

[11] R. Eramo, A. Pierantonio, J. R. Romero, and A. Vallecillo. Change management in multi-viewpoint systems using ASP. In *Proc. of WODPEC 2008*, Munich, Germany, Sept. 2008.

[12] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple perspectives in systems development. *SEKE journal*, 2(1):31–58, 1992.

[13] I. García-Magariño, J. J. Gómez-Sanz, and R. Fuentes-Fernández. INGENIAS development assisted with model transformation by-example: A practical case. In *Proc. of PAAMS'09*, 2009.

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

[15] J. H. Hausmann, R. Heckel, and S. Sauer. Extended model relations with graphical consistency conditions. In *Blekinge Institute of Technology*, pages 61–74, 2002.

[16] IEEE Std. 1471. *Recommended Practice for Architectural Description of Software-Intensive Systems*, 2000.

[17] IKV++ Technologies AG. medini QVT Project. `http://projects.ikv.de/qvt/`.

[18] 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.

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

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

[21] ISO/IEC 42010. *Systems and software engineering – Architectural description*, 2008.

[22] M. Kessentini, H. Sahraoui, and M. Boukadoum. Model transformation as an optimization problem. In *Proc. of MoDELS 2008*, number 5301 in LNCS, pages 159–173. Springer, 2008.

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

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

[25] P. Linington. Black Cats and Coloured Birds What do Viewpoint Correspondences Do? In *Proc. of WODPEC 2007*, Maryland, USA, Oct. 2007.

[26] OMG. *Object Constraint Language (OCL) 2.0*. OMG, May 2006. ptc/06-05-01.

[27] OMG. *Unified Modeling Language 2.1.1 Superstructure Specification*. OMG, Feb. 2007. formal/07-02-05.

[28] OMG. *Meta Object Facility Query / View / Transformation (MOF QVT) Specification 1.0*, Apr. 2008. formal/08-04-03.

[29] OMG. *Unified Profile for the DoDAF and the MoDAF Specification*, 2008. `http://www.updm.com`.

[30] J. R. Romero and J. I. Jaen. MagicDraw plugin for RM-ODP and UML4ODP. `http://www.jrromero.net/tools/mdplugin`, 2009.

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

[32] J. R. Romero and A. Vallecillo. Well-formed rules for viewpoint correspondences specification. In *Proc. of WODPEC 2008*, Munich, Germany, Sept. 2008.

[33] D. Ruiz-Gonzalez, N. Koch, C. Kroiss, J.-R. Romero, and A. Vallecillo. Viewpoint synchronization of UWE models. In *Proc. of MDWE 2009*, June 2009.

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

[35] M. Siikarla, M. Laitkorpi, P. Selonen, and T. Systä. Transformations have to be developed ReST assured. In *Proc. of ICMT'08*, pages 1–15, 2008.

[36] P. Stevens. Bidirectional model transformations in QVT: Semantic issues and open questions. In *Proc. of MoDELS 2007*, number 4735 in LNCS, pages 1–15. Springer, 2007.

[37] R. V. D. Straeten, J. Simmonds, and T. Mens. Detecting inconsistencies between UML models using description logic. In *Proc. of Description Logics (DL 2003)*, 2003.

[38] UK MoD. Architecture Framework (MODAF) M3 — MODAF Meta-Model, May 2007.

[39] D. Varró. Model transformation by example. In *Proc. of MoDELS 2006*, number 4199 in LNCS, pages 410–424. Springer, 2006.

[40] D. Varró and Z. Balogh. Automating model transformation by example using inductive logic programming. In *Proc. of SAC'07*, pages 978–984, 2007.

[41] M. Wimmer, M. Strommer, H. Kargl, and G. Kramler. Towards model transformation generation by-example. In *Proc. of HICSS'07*, 2007.

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