Characterizing Relations between Architectural Views

Nelis Boucké[1], Danny Weyns[1], Rich Hilliard[2],
Tom Holvoet[1], and Alexander Helleboogh[1]

[1] DistriNet Labs, K.U. Leuven, Belgium
{nelis.boucke,danny.weyns,tom.holvoet,
alexander.helleboogh}@cs.kuleuven.be
[2] Consulting software systems architect
r.hilliard@computer.org

Abstract. It is commonly agreed that an architectural description (AD) consists of multiple views. Each view describes the architecture from the perspective of particular stakeholder concerns. Although views are constructed separately, they are related as they describe the same system.

A thorough study of the literature reveals that research on relations between views is fragmented and that a comprehensive study is hampered by an absence of common terminology. This has become apparent in the discussion on inter-view relational concepts in the revision of IEEE 1471 as ISO/IEC 42010 (*Systems and Software Engineering — Architectural Description*).

This paper puts forward a framework that employs a consistent terminology to characterize relations between views. The framework sheds light on the usage, scope and mechanisms for relations, and is illustrated using several representative approaches from the literature. We conclude with a reflection on whether the revision of ISO 42010 aligns with our findings.

Keywords: architectural views, view relations, viewpoint, architectural descriptions, integration of views, consistency, models, IEEE 1471, ISO/IEC 42010.

1 Introduction

The architecture of a software system defines its essential structures, which comprise software elements, the externally visible properties of those elements, the relationships between them [3] and with their environment [21]. It is commonly agreed that an architectural description (AD) consists of multiple views. Views are used to achieve separation of concerns where each view describes the architecture from the perspective of related stakeholder concerns [21]. Although views can be constructed separately, they must be related in that they describe the same system.

Relying on implicit relations, e.g. relating elements having the same name and type, might be sufficient for simple architectures but is insufficient for more complex systems. An important part of the architect's job is to understand, describe and reason about how the different views relate to each other [8,40]. In this paper, we explicitly focus on the relations between architectural views; not on relations between elements within views.

Relations are essential for establishing consistency and for maintaining that consistency over time. Software architects need relations to manage the multitude of views. Developers need relations for an integrated picture of the architecture that is a prerequisite for detailed design and implementation. Other stakeholders need to see how their

R. Morrison, D. Balasubramaniam, and K. Falkner (Eds.): ECSA 2008, LNCS 5292, pp. 66–81, 2008.

concerns are realized and how these realizations relate to concerns of other stakeholders. Finally, relations provide the basis for automation in architectural tools of consistency checking, and integration or synchronization of multiple architectural views during design.

Problem Statement. In the architecture community, there is a common understanding of what an architectural view is. There are several seminal works on views, including Perry and Wolf [36] and Kruchten's 4+1 view model [26]. More recently, multiple views form the basis for approaches such as *Documenting Software Architectures* [8] and *Software Systems Architecture* [40]. Also, the concepts of architectural description and view have been standardized by ANSI, IEEE, ISO and IEC.[1]

Such common understanding however is lacking for relations between views. This lack of consensus became apparent in the recent discussions on the incorporation of relations between views in the ongoing revision of ISO 42010. A thorough study of the literature reveals that research on relations between views is fragmented and hampered by an absence of common terminology. Authors typically compare to approaches with similar purpose (e.g. automatic consistency checks). However, they tend to neglect other relations with similar technical characteristics but devised for another purpose (e.g. enforcing design decisions).

The fragmentation in research also becomes apparent through the myriad of terms in use for related or overlapping concepts. A sample of terms from recent research in this area illustrates the point: [30] uses *constraints, rules, standard constraints, extensions constraints, integration constraints* and *custom constraints*; [18] uses *design constraints, invariants* and *heuristics*; [28] uses *rule* and specializes rules in *constraints* and *obligations*; [14] additionally uses *policy constraints*; [4] uses *viewpoint correspondences*; [8] uses *relations* and *mapping*; [12] uses *refinement* and *overlap relations*, *relationships* and *consistency rules* that apply to the *relations*; [2] uses *relations* and *transformations* for the same thing; [38] uses *boolean rule, general design rule, constraints logic, dependency links, links, design rule* and *transformation rule*; [9] uses *links, relations, rules, correspondences* and *correspondence rules*; [43] uses *traceability links, dependency links, dependency relations, relations, trace relation*; etc. This makes it difficult to characterize and compare approaches for describing relations between views.

Contributions. The revision of ISO 42010 provides an opportunity to offer better guidance to architects for capturing relations between views within an architectural description. This paper contributes a proposed framework that structures approaches for explicit relations between views providing a common ground for relations. The framework is based on a thorough study of the literature and on our experience. The goal is to take a step to disentangle and bring clarification to the work on relations between views. The framework sheds light on the usage, scope and underlying mechanism. Application of the framework is illustrated with several representative approaches from the

[1] The abbreviation *ISO 42010* is used for the published version of ISO/IEC 42010:2007, *System and Software Engineering — Architectural Description* [24]. ISO 42010 is identical in content to ANSI/IEEE Std 1471-2000, *Recommended Practice for Architectural Description of Software-Intensive Systems* [21], and is currently undergoing revision. The abbreviation *42010 WD2* is used for the current working draft of this revision [22].

literature. Based on this, we reflect on whether view relations in the ISO 42010 working draft aligns with the findings in the literature.

Overview. To avoid confusion on terminology issues, section 2 introduces the basic architectural terminology used in this paper. Section 3 proposes the framework to characterize relations between views, and illustrates its use with representative approaches from the literature. Section 4 reflects on the proposal for view correspondences in 42010 WD2 with respect to the proposed framework. Finally, we conclude in section 5.

2 Basic Architectural Concepts

This section introduces the basic architectural concepts and terminology that we will use in the remainder of the paper. There are several known definitions of architecture and architectural views in the literature, the SEI [3,8], Siemens [20], ISO 42010 or RM-ODP [23]. We adopt the conceptual model of ISO 42010, to serve as a consistent set of basic terminology. This does not mean that our scope is limited to the standard; we studied a broad range of approaches in the literature.[2] ISO 42010 has two parts. The first part is a conceptual model for architectural descriptions. The conceptual model introduces and interrelates such concepts as *architectural description*, *concern*, *viewpoint*, *view* and *model*. The second part puts forward required content for any ISO 42010-conformant architectural description, independent of the specific architectural languages in use. Here, we only use the conceptual model.

Figure 1 shows a portion of the ISO 42010 conceptual model relevant for this paper. An *architectural description* (AD) is "a collection of products to document a specific architecture". An AD is organized into one or more *architectural views*, where a view is defined as "a representation of a whole system from the perspective of a related set of concerns". Each view is constructed according to an *architectural viewpoint*, defined as "the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis".

One of ISO 42010's contributions was to explicitly distinguish between view and viewpoint, in this sense: A *viewpoint* is a way of looking at an architecture; the view is the result of looking at a specific system's architecture in this way.[3] First-class viewpoints first appear in Ross' Structured Analysis [39] and are elaborated upon by

[2] In 2007, IEEE 1471 was adopted by ISO as ISO 42010. At present, IEEE and ISO are jointly revising the standard.

[3] Not every architectural approach makes an explicit distinction between view and viewpoint. The term *viewpoint* appears in ISO RM-ODP [23], in a very similar fashion; although *viewpoint specification* is employed where IEEE 1471 uses *view*. In RM-ODP a *viewpoint specification* is defined as "the application of a viewpoint to a specific system". The book *Documenting Software Architectures* (DSA) [8] introduces *viewtypes* as "a viewtype defines the element types and relation types used to describe the architecture of a software system from a particular perspective." DSA further proposes there are three viewtypes: the module viewtype, the component-and-connector viewtype, and the allocation viewtype. In the present framework, these could be considered a three-way classification of viewpoints, in terms of their representational mechanisms (element and relation types).

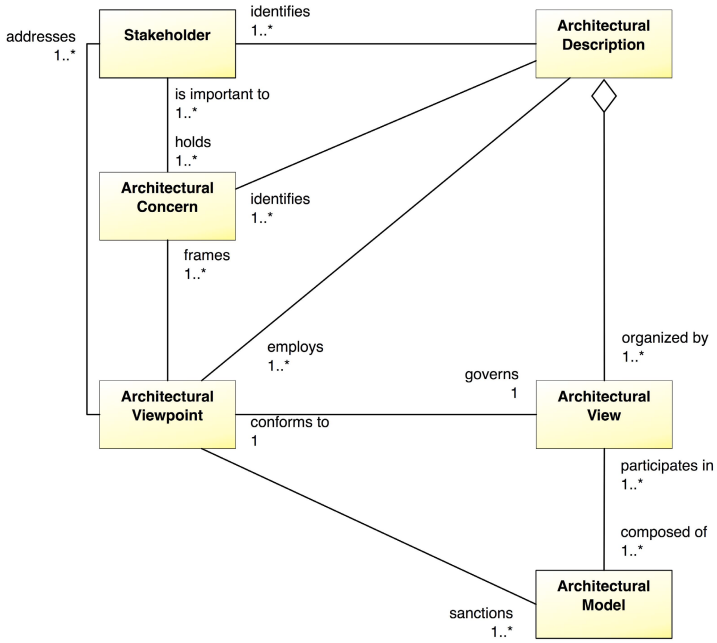

Fig. 1. A portion of the ISO 42010 conceptual model

Finklestein *et al.* [16]. In ISO 42010, viewpoints are intended to provide a representational approach for addressing specific architectural concerns: "those interests which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders". A system stakeholder is any "individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system".[4] Each architectural view is composed of one or more *architectural models*. Each model contains a concrete description of architectural elements and relations, and obeys the conventions of the governing viewpoint with respect to the viewpoint language(s) and model type(s) employed therein. When a view is composed of multiple models, relations can arise between models within a view as well as between views.

The authors of ISO 42010 recognized the issue of view consistency, but did not specify a mechanism for enforcing that consistency except to require the recording any known inconsistencies. We return to the current proposal for introducing relations into the ISO 42010 working draft below section 4.

3 A Framework for Characterizing Relations between Views

Starting from a thorough study of literature and our experiences, we have identified several criteria to characterize relations between views. Each of the criteria emphasizes a particular aspect of view relations. Together, the criteria make up a framework that

[4] The standard uses *concern* in the sense of E. Dijkstra's *separation of concerns*.

Usage				Scope				Mechanism		
Consistency Checking	Composition	Tracing	Model Transformation	Intra vs. Inter Model Type	Level of Detail	Horizontal vs. Vertical	Metamodel vs. Model	Direct References	Tuples	Expression language

Fig. 2. Overview of the criteria for the framework in three orthogonal dimensions

allows structuring and comparing approaches for relations between views. Figure 2 shows an overview of the criteria. The criteria are grouped along three dimensions: Usage, Scope, and Mechanism.

In this section, we describe the criteria and discuss representative examples from the literature. It is not our ambition to be exhaustive in all possible criteria or references, but rather to offer an initial impetus towards defining a framework for characterizing relations between views. At the end of this section, we discuss possible extensions of the framework.

3.1 Usage

We have identified four main use cases for relations between views: (1) consistency checking; (2) composition; (3) tracing; and (4) model transformation. There is no one-to-one mapping between particular types of relations and their use; relations can be used for different purposes. Some of the use cases are not goals in themselves but can serve a further purpose. For example, tracing is typically used for backtracking of decisions and to allow easier changes to the architecture.

Consistency Checking. Consistency checking is about determining whether the information in several views does not conflict. We discuss three example uses of relations between views for consistency checking: general-purpose consistency checking, design constraint checking, and consistency checking of service compositions.

Nentwich et al. [30] put forward a general-purpose approach for automatic consistency checking of heterogenous and distributed software engineering documents. The approach uses constraints underpinned by a first-order logic specification. The associated xlinkit tool generates "inconsistency relations" between elements. A software architect can use the inconsistency relations to identify the elements that cause inconsistencies between several views,[5] and possibly alter particular elements to resolve inconsistencies. xlinkit is not limited to finding inconsistencies between architectural views. The approach has also been used for the identification of inconsistencies in distributed product catalogs [29], requirement specifications [31], UML diagrams [29], web service compositions [14].

Garlan et al. [18] use the Armani language to express design constraints in Acme and the associated AcmeStudio tool. Acme is a general-purpose, component and connector (C&C)-based ADL with particular support for architectural styles. Design constraints

[5] The authors do not distinguish between viewpoints and views. A view as defined in section 2 is called the viewpoint specification in their approach.

can be defined for architectural styles that impose restrictions on how an architectural design is permitted to evolve over time. For example, for a layered architectural style, the constraints will describe that a higher layer is allowed to use a lower layer but not the other way around.

Dijkman and Dumas [11] propose an approach for consistency checking in service composition between four interrelated views.[5] namely interface behavior, provider behavior, choreography, and orchestration. The views are formalized using Petri nets. By specifying relations between the views, the approach enables static verification of a web service composition.

Other examples of consistency checking are described by Boiten et al. [4] and Dijkman et al. [12] in the context of RM-ODP [23], Fradet et al. [17] use graphs and constraint expressions on these graphs, and Radjenovic and Paige [38] developed an ADL and consistency checking in the context of dependable systems. Radjenovic and Paige emphasize that to reach a strong sense of consistency between views, the number and complexity of the constraints increases significantly.

Other approaches deal with this issue of consistency in ADs without first-class relations. The ArchStudio tool for xADL [10] supports plug-ins to analyze an AD. The plug-ins can be used for automatic consistency checking over views. In this case, relations are programmed in the component. In *Software Systems Architecture* [40], Rozanski and Woods provide an extensive checklist of possible relations between different types of views. The list allows software architects to check whether an AD is consistent; however, the relations between views are described only informally.

Composition. Composition of views (sometimes also referred to "merging" or "integration" of views) allows the integration of information from several views. Composed views are useful to get a unified perspective, to understand the interactions between elements from different views, and to perform various types of analysis.

In his doctoral thesis, Egyed [15] presents a framework for integrating multiple heterogenous views. This framework exists of a set of integration activities, including the identification and cross-referencing of related model elements that describe overlapping and thus redundant pieces of information (called mapping). These relations are then used during the differentiation and transformation activities for integrating views with each other. The integrated views serve as a basis for several types of analysis, such as checking consistency.

Boucké et al. [6] introduce three types of relations (unification, refinement and composition) between structural views and demonstrate how these relations allow composition of structural views. Following the 'Having divided to conquer, we must reunite to rule' philosophy [25], the authors state that views can be used to describe concerns, but that relations and the associated composition are essential to bring the views together. Tool-supported composition allows one to easily generate overlays to understand and reason about the integration of several architectural views. The authors have integrated the relations in xADL and the ArchStudio tool suite [5].

Most composition approaches do not distinguish relations from compositions but program the relations directly in a composition algorithm or in composition operators. Abi-Antoun et al. [1] describe an algorithm and associated tool for differentiating and merging C&C views. The authors argue that architects often face the problem of

reconciling different versions of architectural models, e.g. by using specific information from two versions to produce a new version that includes changes from both earlier versions. Sabetzadeh et al. [41] envision the use of explicit relations for merging views. Giese and Vilbig [19] present an approach to compose the behavior of several C&C models. The authors mention that views are related, but relations are not first class—instead programmed within the composition algorithm.

Tracing. Tekinerdoğan et al. [43] document explicit trace relations between architectural concerns, the architectural elements that address the concerns, and between architectural elements in general. In case the architectural elements related to a particular concern that are spread across different architectural views, the proposed trace relations span multiple architectural views. With respect to evolution of the software, one can follow the trace links to update and synchronize architectural views, keeping the software architecture consistent.

Model Transformation. A model transformation takes as input a model conforming to a given metamodel and produces as output another model conforming to a given metamodel. Model transformation is central to the domain of model driven architecture (MDA) [32]. We discuss two approaches using relations in the context of MDA: an approach using a refinement relation and another approach using relations for automatic transformation.[6]

Architectural stratification proposed by Atkinson and Kühne [2] combines the strength of separation of concerns and aspect-orientation with component-based frameworks and model-driven architecture. The goal of architectural stratification is to relate different architectural strata[7] so that they best represent a system's crosscutting concerns. Each stratum represents a software architecture on a certain level of abstraction. The authors use stepwise refinement to relate the strata. In each step, a refinement transformation is applied that refines connectors introducing a particular concern. Relations are thus defined as refinement transformations.

Cordero and Salavert [9] use relations on the metamodel level with the goal of automated transformation of architectural models. Example relations are: 'each module is related to a component' and 'each uses relation between modules to a connector'. The combination between the module view and the relations on the metamodel level allows one to automatically generate a component and connector view. The approach proposed by Dijkman et al. [13] is similar, but focuses on the RM-ODP views.

Recently, a standard for model transformation has been defined by the Object Management Group, the Meta Object Facility Query/View/Transformation Specification (QVT) [34]. QVT advocates explicit specification of relations before performing transformations. QVT provides a conceptual model and basic architecture for model-driven transformation tools. Because the standard is very recent, efforts are still on the way to conform approaches and tools to the QVT model.

[6] Alternatively model transformations could also be considered as a mechanism, where relations are implemented by transformations between models. We placed it under usage, as model transformations typically also embodies a process, and it is in this model transformation process that relations are used.

[7] An *architectural stratum* is a kind of architectural model as defined in section 2.

3.2 Scope

Scope refers to the extent or range of view relations. From the literature study, we identified four criteria for the scope of a relation: (1) intra vs. inter model type; (2) level of detail; (3) metamodel vs. model; and (4) horizontal vs. vertical.

Intra vs. Inter Model Type. Intra model type relations are relations between the same type of models. An example is a relation between two C&C models. Inter model type relations are relations that involve models of different types. For example, relations between a C&C model and a statechart model.

Several approaches we discussed in previous sections define relations between C&C models only. Examples are Garlan et al. [18], Atkinson and Kühne [2], and Boucké et al. [6]. Dashofy et al. [10] support inter model relations, in particular relations between types, structural elements, and component instances. xlinkit constraints [30] can be defined on any XML document, and as such also supports inter model relations.

Level of Detail. Relations can be described between complete views, between models, and between architectural elements inside views.

Clements et al. [8] use sibling and child relations to relate architectural models.[8] Sibling models document different parts of the same system. These models form a mosaic of the whole view, as if each partial view were a photograph taken by a camera that panned and tilted across the entire view. Child models document the same part of the system but in greater details. This is a coarse-grained kind of refinement. Sibling and parent/child relations are specified at the level of complete models; they do not allow specifying details about which particular architectural elements are related.

Architectural unification proposed by Melton and Garlan [27] supports fine grained specification of relations, up to the level of component interfaces and properties that are associated with the components.

Horizontal vs. Vertical Relations. The term *horizontal* is used for relations between views at the same level of abstraction. *Vertical* relations are either relations between views at different levels of abstraction (such as refinements) or relations with other representations (such as requirements, detailed design or even implementation). The terms *intra phase* and *inter phase* are also used in this context, for horizontal and vertical, respectively.

The approach to architectural stratification [2] is a good example of the use of vertical relations. The strata correspond to models at different levels of abstraction and are step-by-step refined via transformation. Both Nentwich et al. [30] and Radjenovic and Paige [38] propose approaches with explicit support for both horizontal and vertical relations. The latter propose transformation rules to capture relations between subsequent development stages. Muskens et al. [28] is a language-neutral approach that addresses intra and inter phase view relations.

Most other approaches do not define an explicit process to use relations in a vertical way. For example, Armani constraint relations are typically used to specify invariants

[8] The authors use the term 'view packets'; in the terminology of section 2, these can be considred architectural models.

over several models at one level. However, it is possible to define a process on top of Acme and Armani that uses constraints to support things like refinement.

Metamodel vs. Model. A metamodel is an explicit model of the constructs and rules to build specific models within a domain of interest. In the terminology of section 2, the metamodel refers to the part of the viewpoint language that defines an individual model type. View relations may be stated with reference to a metamodel or between metamodels.

A well-known example of metamodel relations are the constraints in the superstructure of the UML 2.0 definition [33]. The superstructure defines the language elements of UML 2.0 models and the constraints on how those language elements can be used within diagrams as well as across multiple related models.

Cordero and Salavert [9] require that each module is related with one component, and that each usage association is related with one connector. This enables automatic transformations between the models (section 3.1).

xlinkit constraints include expressions that reference specific points in an XML document [30]. Expressions can refer to elements at the model level as well as the metamodel level.

3.3 Mechanism

Support for relations between views requires constructs in the AD to represent those relations. We have identified three classes of mechanisms from the literature to describe relations between views, namely: (1) direct references; (2) tuples; and (3) expression languages.

Direct References. Elements from one view can refer directly to elements of another view. In this case, the description of the relations between architectural elements of different views is mingled with the view descriptions.

Dashofy et al. [10] use direct references between the architectural views of xADL. xADL allows several views, including a view specifying component types (types view), views showing the structure of the system by connecting component types (structural view) and views showing component instances (instance view). Components in a structural view can refer to their types. Component instances can refer to the structural components they adhere to and instances can refer to an internal structure (refinement) described in a structural view.

AADL [42] uses packages to structure architectural documentation. Packages group architectural elements (component types and instances) into logical blocks, so that there is a clear connection between the concepts of package and architectural view. The relations between the packages are directly described in the packages; i.e. a component instance can directly refer a component type in another package.

Tuples. In mathematics, a relation over sets is defined as a subset of the Cartesian product of the sets. Elements in the relational set are called tuples. Relations modelled as tuples are typically complex in the context of ADs. Architectural views typically contain several architectural models, each model defining several types of elements

with possibly complex internal details. This may require one to annotate the tuple with more details of how the elements are related.

Documenting Software Architecture [8] introduces a 'mapping table' to describe the relations between views, as a part of the information "beyond" the views. Mapping tables define a set of tuples of elements from different views (one-to-many, many-to-many, many-to-one). Each table entry is annotated with a textual description to indicate whether the correspondence is partial and to provide additional details of the relation (such as corresponding interfaces). Tables can be used for any relation based on tuples.

Boiten et al. [4] also define relations between elements in a table, and annotate each entry with the type and the level of detail of the relation. The authors define relations between two RM-ODP Engineering views, and between the Engineering view and Computation view. Relations have a formal underpinning in ObjectZ. The details of the relations sometimes contain expressions as defined in the next section.

Expression Language. In general, an expression in mathematics is a combination of names and values, operators, grouping symbols (such as brackets), and possibly variables (free and bound) arranged in a meaningful way. Expressions containing variables may use quantifiers (such as $\forall$ and $\exists$). An expression language defines which expressions are well-formed, and therefore can be used and meaningfully interpreted. In the context of ADs, the expressions impose constraints or rules over (sets of) architectural elements. The complexity of the expression language may vary widely, depending on factors such as the formal system (e.g. first-order logic) on which it is based, the underlying viewpoint languages being related, etc.

The previously mentioned xlinkit tool [30] for automatic consistency checking between architectural views uses constraints. The tool processes XML documents, using a formal underpinning based on an extension of first-order logic. It is possible to describe things such as:

$$\forall c \in \text{'components'}\ (\exists m \in \text{'module'}\ (c.modulename == module.name))$$

The expressions 'components' and 'module' are XPath [44] expressions that select the sets of elements involved in the relation as a tree path in the XML document.

Muskens et al. [28] introduce a general approach for detecting inconsistencies between different views based on relational partition algebra. Its expression language includes named relations and operations such as inclusion, composition, intersection, union, inverse and transitive closure on those relations, but is quantifier-free. The approach introduces the interesting notions of prevailing vs subordinate views, that can be used in horizontal or vertical view relations. The distinction is used, for example, to report violations in the subordinate view, taking the content of the prevailing view as fixed. An example is a constraint between a message sequence diagram and a class diagram that requires that the dependencies between classes implied by the message sequence diagram are present in the class diagram. Checking this constraint is not trivial, since inheritance must be taken into account:

$$((CALLER; CALLEE^{-1}) \uparrow TYPE) \subseteq (DEPENDENCY \downarrow INHERITANCE*)$$

This rule states that all calls between caller and callee objects, lifted ($\uparrow$) to the types (leading to dependencies between the types of these objects), must be a subset of the

dependency set of the class diagram, lowered (↓) to inheritance (taking subclasses into account). The upwards arrow and downwards arrow are algebraic functions for respectively lifting or lowering the level of abstraction.

3.4 Discussion

Starting from a thorough study of the literature and our experience, we have proposed a framework for analyzing approaches to relations between views in three dimensions: usage, scope and mechanism. The illustrations from the literature provide a first indication of its usefulness, but the practical value of the framework for software architects remains to be proven. Although we believe that the framework adequately captures the existing work on relations between views, we do not claim that the framework is complete. One may discover additional use cases, and/or refine or extend the current dimensions.

An interesting criterion to add could be the way relations are formalized, and what underlying mechanism is used to support those relations. The nature of the formalization has implications on what analyses can be performed and what outcomes or results can be generated from those analyses (e.g. proofs of consistency, counterexamples, etc.). The underlying mechanism supporting relations can vary widely. Some approaches use first-order logic and a theorem prover to search for inconsistencies. Some establish consistency of two representations by formalizing them in ObjectZ and finding a common refinement.

4 Reflection on Relations between Views in ISO 42010

The revision of ISO 42010 provides an opportunity to capture common concepts and terminology in the area of views and relations between views. We first explain the proposal for relations in the current working draft of ISO 42010 (42010 WD2). Next, we compare the 42010 WD2 proposal with our findings about the literature, embodied in the framework outlined in the previous section.

4.1 Relations between Views in 42010 WD2

The working draft proposes a new concept: *view correspondence* (VC). A VC records a relation between two architectural views to capture: a consistency relation, a traceability relation, a constraint or obligation of one view upon another. Mathematically, a VC is a binary relation. The intent is that an AD might include several VCs to express one or more relations among its views.

> *Example*: Consider two views of a system, S, a hardware view, $HW(S)$, and a software component view, $SC(S)$. If $SC(S)$ includes software components, $e_1, \ldots e_4$, and $HW(S)$ includes hardware platforms, $p_1, \ldots p_4$, a view correspondence between $HW(S)$ and $SC(S)$, specifying which components execute on which platforms, might be:
>
> $$ExecutesOn = \{(c_1, p_1), (c_1, p_4), (c_2, p_2), (c_2, p_3), (c_3, p_3), (c_4, p_4)\}$$

In the context of the framework, this is a tuple-based specification between different types of models (inter model type). The level of detail is architectural elements: each item in the relation is a complete component or platform. $ExecutesOn$ describes a horizontal relation between two concrete models.

In addition to VCs, 42010 WD2 introduces *viewpoint correspondence rules* (VCRs). A VCR expresses a required relation between two architectural viewpoints and is realized by VCs on views resulting from the application of those viewpoints within an AD.

> *Example*: Every software component, c_i, as defined by a software component viewpoint applied to system S, $SC(S)$, must execute on one or more platforms, p_j, as defined by a hardware viewpoint applied to that same system S, $HW(S)$.
>
> $$ExecuteOnRule = \forall c_i \in SC(S) : \exists p_a \in HW(S) : (c_i, p_a) \in ExecutesOn$$

In the context of the framework, this is an expression with quantifiers in an expression language between different types of models (inter model type). The level of detail is architectural view elements: each variable is a complete component or platform. The rule describes a horizontal relation between two concrete models.
A VCR imposes the following requirements on VCs (in 42010 WD2):

- For each VCR that applies to an AD, there shall be a VC identified.
- A VCR *holds* in an AD if its associated VC can be shown to satisfy the rule.
- A VCR *is violated* in an AD if its associated VC can be shown not to satisfy the rule.

4.2 Comparison to Framework

Since 42010 WD2 is still a working draft, the effectiveness of the view correspondence proposal remains to be proven. In this section we compare the proposal with our findings in the literature, embodied in the framework of section 3. The discussion is structured according to the criteria of the framework.

Usage. 42010 WD2 contains an open list of possible uses, but stays neutral to what purpose relations are used, which is consistent with the method-neutral stance of the standard. However, a number of limitations which we have identified in the context of scope may imply restrictions on the possible use of relations. We describe these limitations next.

Scope: Model vs. Metamodel. There is a similarity between model and metamodel on the one hand and the concepts of VC and VCR on the other hand.
A VC is a relation between two views, often expressed as a relation at the model level. The similarity between a VCR and a metamodel relation is less obvious. Metamodels (or model types) are part of the viewpoint language and as such not explicitly represented in ISO 42010. A VCR is defined between viewpoints, expressed as a relation between viewpoint languages. From this point of view, a VCR can be considered as a relation at the metamodel level.

Notice that VCR and VC are tightly coupled concepts. Such coupling is typically less explicit between metamodel relations and model relations in the literature.

Scope: Inter vs. Intra. The use of inter vs. intra model type relations is less clear in 42010 WD2. In ISO 42010 a view is a representation of the *whole* system with respect to some concerns. Intra model type relations would be part of the same (viewpoint) language, and would therefore typically be models within the same view. Inter model type relations would be part of different (viewpoint) languages, which may or may not be models in the same view.

This contrasts with our observation of the literature: intra model relations are typically not limited to models that are in a single view. For example, it is quite common to have multiple C&C models in different architectural views.

Scope: Level of detail. VCs and VCRs are not restricted to a particular level of detail, and as such cover the different levels of detail of relations that we have seen in the literature.

Scope: Horizontal vs. Vertical. 42010 WD2 does not state anything about horizontal and vertical relations. The concepts of the standard can be applied in a horizontal or vertical manner.

Mechanism. A standard should be mechanism-neutral. 42010 WD2 does not make explicit statements about the mechanism to be used for view correspondences or viewpoint correspondence rules.

We give a side remark on the term "rule" in VCR, which may be confusing. The idea is that each VCR imposes an obligation on views that must be demonstrated by a VC. The term rule is often used for a specific mechanism to specify relations, suggesting the use of a mathematical expression in the sense of section 3.1. A VCR can just as well be represented as a direct reference or a tuple between language elements.

We have two other small remarks on the terminology. Firstly, the term 'correspondence' is used in the context of RM-ODP, but outside this scope the more neutral term 'relation' seems to be used more often. Secondly, the term *viewpoint* correspondence rule could lead to confusion, since the obligation is imposed on a view. *View* correspondence rule seems closer to the intent of the proposal.

In summary, the 42010 WD2 proposal largely aligns with our observations of the literature. Yet, there is some unclarity with respect to: (1) the advice that intra model relations are typically within a single view; (2) the role of model types; and (3) the terminology of view correspondences and viewpoint correspondence rules. It is hoped these can be clarified upon in future revision drafts.

5 Conclusion

Views and view relations have been studied for a long time. However, existing work on view relations is fragmented. The framework presented in this paper shows that there is a common ground for relations. There are strong arguments for making relations first-class concepts in ADs, treating them on equal terms as architectural views. As soon as an AD contains multiple views, these views are related since they describe the same system. Making the relations explicit improves the clarity of the architectural documentation. It forms the basis for consistency checking, for automatic analysis and

verification of quality attributes and system wide properties, for tracing design decisions, etc.

An important observation is that existing ADLs, such as AADL, xADL, Acme, π-ADL [35], Fractal ADL [7] and AO-ADL [37] offer support for multiple types of architectural elements, but do not offer first-class support for architectural views in a way advocated by ISO 42010. ADLs lack facilities for *specifying* and *relating* several architectural views that cope with a diversity of architectural concerns. To the best of our knowledge, the AIM ADL for embedded systems [38] is the only notable exception. Imperative to exploit such view-based ADLs will be tool support. A tool can interactively suggest view relations based on particular heuristics, such as similar names and similar architectural patterns. Visual editors can simplify the specification of relations between views. On the fly generation and visualization of overlay views, or highlighting the elements involved in a relation, can improve the understanding and use of relations between views.

As a closing remark, a first-class concept of relations is just the start. A lot of work must be done to concerning practical problems like conflicts between views, integration of views, comparisons between different approaches in modeling the same views, and how to enforce consistency amongst views.

Acknowledgement

We are grateful to Christina von Flach, Peter Eeles, Bedir Tekinerdoğan, Hasan Sozer, Tomi Männistö, Thorsten Keuler and the other attendees of the Birds-of-a-Feather session on Relations between Views at WICSA 2008 for the interesting discussions. We also express our appreciation for the valuable input and feedback from Dimitri Van Landuyt, John Klein, Rich Paige and Steven Op de beeck.

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U.Leuven. Nelis is supported by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen). Danny is supported by the Foundation for Scientific Research in Flanders (FWO-Vlaanderen).

References

1. Abi-Antoun, M., Aldrich, J., Nahas, N., Schmerl, B., Garlan, D.: Differencing and merging of architectural views. Automated Software Engineering 15(1), 35–74 (2008)
2. Atkinson, C., Kühne, T.: Aspect-oriented development with stratified frameworks. IEEE Software 20(1), 81–89 (2003)
3. Bass, L., Clements, P., Kazman, R.: Software Architectures in Practice, 2nd edn. Addison-Wesley, Reading (2003)
4. Boiten, E., Bowman, H., Derrick, J., Linington, P., Steen, M.: Viewpoint consistency in ODP. Computer Networks 34(3), 503–537 (2000)
5. Boucké, N.: xADLComposition: a tool for view composition in xADL, `http://www.cs.kuleuven.be/~nelis/xADLComposition`
6. Boucké, N., Holvoet, T.: View composition in multi-agent architectures. Special issue on Multiagent systems and software architecture, International Journal of Agent-Oriented Software Engineering (IJAOSE) 2(2), 3–33 (2008)

7. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.B.: The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. Softw. Pract. Exper. 36(11-12), 1257–1284 (2006)
8. Clements, P., Bachman, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: Documenting Software Architectures, Views and Beyond. Addison-Wesley, Reading (2003)
9. Cordero, R.L., Salavert, I.R.: Relating software architecture views by using MDA. In: Gervasi, O., Gavrilova, M.L. (eds.) ICCSA 2007, Part III. LNCS, vol. 4707, pp. 104–114. Springer, Heidelberg (2007)
10. Dashofy, E., van der Hoek, A., Taylor, R.: A comprehensive approach for the development of modular software architecture description languages. ACM Transactions on Software Engineering and Methodology (TOSEM) 14(2), 199–245 (2005)
11. Dijkman, R.M., Dumas, M.: Service-oriented design: a multi-viewpoint approach. International journal of cooperative information systems 13(4), 337–368 (2004)
12. Dijkman, R.M., Quartel, D., van Sinderen, M.J.: Consistency in multi-viewpoint design of enterprise information systems. Information and Software Technology (2007)
13. Dijkman, R.M., Quartel, D.A.C., Pires, L.F., van Sinderen, M.J.: An approach to relate viewpoints and modeling languages. In: Proceedings. Seventh IEEE International Enterprise Distributed Object Computing Conference, pp. 14–27 (2003)
14. Dingwall-Smith, A., Finkelstein, A.: Checking complex compositions of web services against policy constraints. In: MSVVEIS, pp. 94–103. INSTICC PRESS (2007)
15. Egyed, A.: Heterogeneous view integration and its automation. PhD thesis, Los Angeles, CA, USA, Adviser-Barry William Boehm (2000)
16. Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M.: Viewpoints: a framework for integrating multiple perspectives in system development. International Journal of Software Engineering and Knowledge Engineering 2(1), 31–57 (1992)
17. Fradet, P., Le Métayer, D., Périn, M.: Consistency checking for multiple view software architectures. SIGSOFT Softw. Eng. Notes 24(6), 410–428 (1999)
18. Garlan, D., Monroe, R.T., Wile, D.: ACME: Architectural description of component-based systems. In: Foundations of Component-Based Systems. Cambridge University Press, Cambridge (2000)
19. Giese, H., Vilbig, A.: Separation of non-orthogonal concerns in software architecture and design. Software and Systems Modeling 5(2), 136–169 (2006)
20. Hofmeister, C., Nord, R., Soni, D.: Applied software architecture. Addison-Wesley Longman Publishing Co., Boston (2000)
21. IEEE1471. Recommended practice for architectural description of software-intensive systems (ANSI/IEEE-Std-1471) (September 2000)
22. ISO. Second working draft of Systems and Software Engineering – Architectural Description (ISO/IEC WD2 42010). Working document: ISO/IEC JTC 1/SC 7 N 000
23. ISO. ISO/IEC 10746-2 Information Technology – Open Distributed Processing – Reference Model: Foundations (September 1996)
24. ISO. ISO/IEC 42010 Systems and Software Engineering – Architectural Description (July 2007)
25. Jackson, M.A.: Some complexities in computer-based systems and their implications for system development. In: Proceedings of Comp. Euro. 1990. IEEE Computer Society Press, Los Alamitos (1990)
26. Kruchten, P.: The 4+1 view model of architecture. IEEE Software 12(6), 42–50 (1995)
27. Melton, R., Garlan, D.: Architectural unification. In: CASCON 1997: Proceedings of the conference of the Centre for Advanced Studies on Collaborative research, p. 18 (1997)
28. Muskens, J., Bril, R.J., Chaudron, M.R.V.: Generalizing consistency checking between software views. In: WICSA 2005: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture, pp. 169–180. IEEE Computer Society, Los Alamitos (2005)

29. Nentwich, C., Capra, L., Emmerich, W., Finkelstein, A.: xlinkit: a consistency checking and smart link generation service. ACM Trans. Inter. Tech. 2(2), 151–185 (2002)
30. Nentwich, C., Emmerich, W., Finkelstein, A., Ellmer, E.: Flexible consistency checking. ACM Trans. Softw. Eng. Methodol. 12(1), 28–63 (2003)
31. Nuseibeh, B., Kramer, J., Finkelstein, A.: Expressing the relationships between multiple views in requirements specification. In: International Conference on Software Engineering, pp. 187–196 (1993)
32. OMG. Model Driven Architecture (MDA)
33. OMG. Unified Modeling Language 2.0: Superstructure (August 2004)
34. OMG. Meta Object Facility 2.0: Query/View/Transformation Specification (August 2007)
35. Oquendo, F.: Pi-adl: an architecture description language based on the higher-order typed pi-calculus for specifying dynamic and mobile software architectures. SIGSOFT Softw. Eng. Notes 29(3), 1–14 (2004)
36. Perry, D.E., Wolf, A.L.: Foundations for the study of software architecture. SIGSOFT Softw. Eng. Notes 17(4), 40–52 (1992)
37. Pinto, M., Fuentes, L.: Ao-adl: An adl for describing aspect-oriented architectures. In: Moreira, A., Grundy, J. (eds.) Early Aspects Workshop 2007 and EACSL 2007. LNCS, vol. 4765, pp. 94–114. Springer, Heidelberg (2007)
38. Radjenovic, A., Paige, R.F.: The role of dependency links in ensuring architectural view consistency. In: WICSA 2008: Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), pp. 199–208 (2008)
39. Ross, D.T.: Structured Analysis (SA): a language for communicating ideas. IEEE Transactions on Software Engineering SE-3(1), 16–34 (1977)
40. Rozanski, N., Woods, E.: Software Systems Architecture. Addison-Wesley, Reading (2005)
41. Sabetzadeh, M., Nejati, S., Easterbrook, S., Chechik, M.: A relationship-driven approach to view merging. SIGSOFT Softw. Eng. Notes 31(6), 1–2 (2006)
42. SAE: Society of Automotive Engineers. Architecture analysis and design language (AADL)
43. Tekinerdogan, B., Hofmann, C., Aksit, M.: Modeling traceability of concerns for synchronizing architectural views. Journal of Object Technology 6(7), 7–25 (2007)
44. W3C. XML path language (XPath), `http://www.w3.org/TR/xpath`