From Viewpoints and Abstraction Levels in Software Engineering Towards Multi-Viewpoints/Multi-Hierarchy in Software Architecture.

Ahmad Kheir, Hala Naja, Mourad Chabane Oussalah, Kifah Tout

▶ **To cite this version:**

Ahmad Kheir, Hala Naja, Mourad Chabane Oussalah, Kifah Tout. From Viewpoints and Abstraction Levels in Software Engineering Towards Multi-Viewpoints/Multi-Hierarchy in Software Architecture.. The Eighth International Conference on Software Engineering Advances, Oct 2013, Venice, Italy. pp.478. hal-01006222

HAL Id: hal-01006222
https://hal.science/hal-01006222v1

Submitted on 3 Jul 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

From Viewpoints and Abstraction Levels in Software Engineering Towards Multi-Viewpoints/Multi-Hierarchy in Software Architecture.

Ahmad KHEIR[1,2], Hala NAJA[1], Mourad OUSSALAH[2] and Kifah TOUT[1]
[1]*LaMA Laboratory, EDST - AZM Center For Research - Lebanese University, Tripoli, Lebanon*
[2]*LINA Laboratory, Nantes University, Nantes, France*
{*ahmad.elkheir, Mourad.oussalah*}*@Univ-nantes.fr,* {*hala.naja, kifah.tout*}*@ul.edu.lb*

Keywords: Software Architecture: Viewpoints: Views: Abstraction Levels: Dependency: Consistency Rules.

Abstract: Viewpoints concept could be considered one of the major concepts introduced in the software engineering domain in order to enhance the architectural organization of complex systems by separating its concerns. Despite his ultimate importance, this concept must be evolved and hierarchized in order to allow the jump of software architectures field from its current range of complexity coverage and resolution to a new range more secure, more standardized and more appropriate with the current industrial needs.
This paper begins with a survey of the role and usage of the viewpoints, and the hierarchy definition by abstraction levels concepts. Then presents a small but complete analysis of the related works and their limitations, in order to conclude with a proposition of current work on a multi-viewpoints and multi-abstraction levels software architecture.

1 INTRODUCTION

Complex systems always include a large amount of requirements, associated to different categories of stakeholders that must be managed in coherent software architecture. Furthermore, each stakeholder adopts his own look to the system, represented via a specific viewpoint that is in most cases complex and hard to be covered entirely in the system architecture. Informally, a view is an architectural design of a stakeholder's viewpoint that covers delicately all its interests and required functionalities, and represents them formally in a structured architecture. An abstraction level represents a level of details defined in order to treat the extra complexities retained after decomposing a system into multiple viewpoints, by relegating some viewpoints' details which appear to be irrelevant in the first stages of the modeling process to lower abstraction levels, so the architect could zoom into a specific viewpoint to the desired level of details. This paper presents, in sections 2 and 3, a survey dealing with the evolution of role and usage of two major concepts in software engineering, which are respectively the views and abstraction levels. Then an overview of an architectural modeling approach called *MoVAL* is suggested in section 4. In *MoVal*, both views and abstraction levels concepts are integrated allowing the architect to build robustly a multi-views and multi-granularities software architectures. Section 5 and 6 presents a comparative analysis and the deducted limitations of current solutions. Section 7 concludes.

The work described in this paper is partially supported by a grant from National Council for Scientific Research (Lebanon).

2 HISTORY OF VIEWS

A view is a formal representation of a stakeholder or a group of stakeholders' viewpoint. This notion has been introduced in four main fields of the software engineering domain, which are: requirements engineering, systems modeling, software architecture and software development.

2.1 Views in Requirements Engineering

One of the first and leading researchers that worked with viewpoint concept in requirements specification was A. Finkelstein in (Finkelstein and Fuks, 1989),

where he and Fuks proposed a formal model aiming to construct system's specifications through a dialog in which the viewpoint negotiate. Robinson also worked on the requirements specification through dialog (Robinson, 1990), in 1990's and presented his tool, *Oz*, but he focused more on the representation of conflicting viewpoints and an automated way to produce resolutions.
Meanwhile, H. Delugach (Delugach, 1990) was working on the representation of each viewpoint's requirements in a conceptual graph in order to merge them by analyzing those conceptual graphs and performing a join so that we obtain a single coherent set of requirements for the target system.
In 1994, B. Nuseibeh et al. have introduced in (Nuseibeh et al., 1994), the relationships that could exist between viewpoints (overlapping, complementary, contradiction) in requirements specification and have proposed a framework to describe those relationships.
Finally, in 1997, Sommerville has contributed in this domain in (Sommerville and Sawyer, 1997) by proposing, *Preview*, a model for requirements discovery, analysis and negotiation.

2.2 Views in Software Systems Modelling

Prior to the incomparable commonness of UML in software systems modeling field, Mullery has developed a method, called CORE, for requirements specification and design in 1979 (Mullery, 1979). CORE is a modeling approach supported by diagrammatic notation which decomposes the modeling process to many steps, and defines for each of them a definition level and a set of viewpoints for which they will model their requirements and establish, in an iterative way, the consistency among them.
In the early 1990s, Booch et al. have developed the *Unified Modeling Language* (UML), which was adopted by the OMG in 1997. In the current version of UML, 13 types of diagrams, where each diagram type has an implicit viewpoint. Several researchers have worked to extend UML model in order to add the concept of dynamic views and viewpoints like in (Ladeira and Cagnin, 2007; Nassar, 2003).
In 2003, a view-based UML extension, VUML (Nassar, 2003), have been introduced to offer a UML modeling tool in which the multiple viewpoints of the system are taken into account in the modeling phase, so they offered a methodology to construct multiple class models one for each viewpoint and merge them to obtain a single VUML model describing all the viewpoints. Then they proposed a code generation (Nassar et al., 2009) and automated composition (Anwar et al., 2011) tools.
Also one of the important works done in this field was (Dijkman et al., 2008) where they presented a framework allowing the architect to model a system from different viewpoints by defining a collection of basic concepts common to all viewpoints and defining for each of them its associated level of abstraction. In this work they focused on the definition of consistency relationships between the views.

2.3 Views in Software Architecture

Software architecture consists on the definition of a structured solution that meets all the system requirements, and represents a complete organization of the entire software system and its inherent software and hardware components. Various models have been proposed of how to create a documentation of those architectures (i.e. an architectural description) by the separation of concerns. Each model describes a set of viewpoints and identifies the set of concerns that each of them address. The concept of Views appears in one of the earliest papers, Perry and Wolf's classic (Perry and Wolf, 1992) on Software Architecture in the early 1990s. In (Sowa and Zachman, 1992), the approach presents an extensive set of constructs called *columns* which is very similar to views. In 1995, Philippe Kruchten proposed four different Views (Kruchten, 1995) of a system and the use of a set of scenarios (use cases) to check their correctness. In (Hilliard, 1999), the author proposed an Architecture Description Framework (ADF) in which views are first-class entities governed by type-like entities called viewpoints characterized in terms of a set of properties pertaining to their application, a viewpoint language. This study was the basis of the ISO/IEC/IEEE 42010 Standard (ISO/IEC/IEEE, 2011) which has formalized concepts used in Software Architecture and brought some standardization of terminology used in this field. It recognized the importance of Views in architectural description and adopted the Viewpoint concept defined earlier in (Hilliard, 1999). In (Clements et al., 2002), 3 viewpoints, called viewtypes, were identified which are: Module, Component-and-Connector and Allocation. In this approach, the author has defined a three-step procedure for choosing the relevant views for a system based on stakeholder concerns.
Next, in 2005, a valuable survey (May, 2005) compared between several view-based models and tried to find out the correspondencies between the different views proposed and the divergence between them. Then it has proposed an optimum framework coverage encompassing a viewpoint set selected from dif-

ferent models with the greatest coverage of the framework concepts. In (Rozanski and Woods, 2011), the authors proposed a Viewpoint Catalogue for Information Systems, extending the 4+1 set of viewpoints identified by Kruchten (Kruchten, 1995) comprising 6 core viewpoints including : the Functional, Information, Concurrency Development, Deployment and Operational.

2.4 Views in Software Development

In Software Development field, many techniques consider that a number of software development *concerns* could not be handled using the modularization boundaries inherent in object-oriented languages and propose new artifacts (beyond method, class or package) to separate new kinds of concerns that tend to be amalgamated in object-oriented paradigms (Mili et al., 2006). In this area, several methods are proposed:

- The Subject-Oriented Programming (SOP) (Ossher and al., 1995) technique addresses the *functional requirements*. It views object oriented applications as the composition of several application slices representing separate functional domains called *subjects*.
- The Aspect-Oriented programming (AOP) technique, such in (Majumdar and Swapan, 2010), addresses the *non-functional requirements*. It defines *aspect* as an implementation of a concern that pertain to several objects in a collaboration.
- The View-Oriented programming technique (Mili and al, 1999) considers each object of an application as a set of core functionalities available, directly or indirectly, to all users of the object, and a set of interfaces specific to particular users.

3 HISTORY OF ABSTRACTION LEVELS

Abstraction level is a core concept in software engineering; it is a way to deal with software systems' complexities giving the architect or the analyst the ability to examine different topics of the system at different levels of details (i.e. abstraction levels) according to the purpose.
Actually there were not so many researches in this domain as it was the case in viewpoints domain, but among the valuable works done in this field (Regnell et al., 1996) could be considered, where authors have proposed a hierarchical modeling approach that consists of three static distinct abstraction levels, which

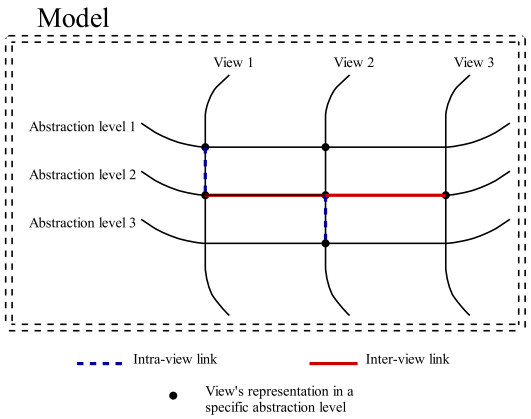

Figure 1: Conceptual matrix of MoVAL model

are the environment level, structural level and event level.
In (Medvidovic et al., 1996), authors have proposed a software architecture's modeling process of component based architectures, composed of four modeling process abstraction levels or, actually, four modeling layers, which are: architectural components specification, components' interfaces definition, architectural description and architectural styles rules.
In (Monperrus et al., 2009), authors have proposed a decomposition approach of meta-model to multiple abstraction levels. In this approach they defined a lower abstraction level as a specialization of the upper abstraction level and proposed splitting rules that are based on the inheritance and specialization relationships of the meta-model.

4 MoVAL APPROACH

MoVAL (Model, View and Abstraction Level based software architecture) is a multi-views/multi-hierarchy software architecture that complies with the IEEE standard (ISO/IEC/IEEE, 2011). The major benefits of MoVAL are: (i) it allows architects to solve more efficiently complexity problems; (ii) it allows them to build a complex, organized and coherent architecture (iii) and enhance the communication and harmony among different stakeholders by giving each of them the appropriate tools to express his interests.
Actually, a MoVAL model could be conceptualized, as shown is figure 1, by a matrix, in which the columns represent the views of the model and the lines represent the abstraction levels of a view.

4.1 Model views

A view of a model in MoVAL, is a representation of the entire system considering a set of the development process aspects and certain problems associated to a specific stakeholder or a group of stakeholders. Moreover, each view of the model is represented in a multi-levels multi-types hierarchy, but this paper will present only the abstraction levels, which are the highest hierarchy levels of a view.

4.2 View's abstraction levels

An abstraction level is the higher level in a view's hierarchy. It represents the level of details considered in certain point of the development process and defines the details that must be considered in this level and what to be relegated to lower levels. Each abstraction level specifies a well-defined diagrammatic notation that must be used to model the associated view considering the current level of details.

4.3 Links

The links are structural elements defined in MoVAL in order to express formally the relations that could exist between different abstraction levels and to conserve, consequently, the model's consistency. Those links are grouped in four categories, but this paper will focus only on two of them:

- **Inter-views link**, defining the relation among a couple of distinct abstraction levels belonging to two different views.
- **Inter-levels link**, defining a similar relation to that defined by the inter-views link, except that the abstraction levels here belong to the same view.

Actually, MoVAL has defined those links formally by attributing for each of them a semantic role that could be a simple connection, composition, expansion, etc. ... Also, MoVAL has created some semantic attributes for each of them in order to express formally all the necessary semantics of a relation between different abstraction levels of a model and guarantee its consistency.

4.4 Case study

To clarify MoVAL concepts, a reduced case study will be introduced in this section. This case study consists on a banking system in which two primary views are considered associated to a hierarchy of one single abstraction level: the Client view and the External Application view. Figures 2 and 3 illustrate those abstraction levels.

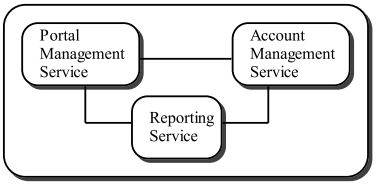

Figure 2: Client view's abstraction level

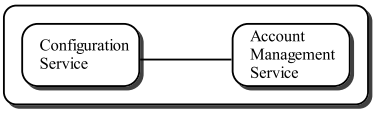

Figure 3: External application view's abstraction level

In the considered portion of the banking system, an inter-views link could be defined between the abstraction levels of those views describing the relation between them and formalizing the fact that the external account management service component of the external application view's abstraction level uses the account management service of the client view's abstraction level.

5 ANALYSIS

Table 1 presents an elicitation of five main characteristics of several approaches sharing similar purposes with MoVAL's approach.

- The inter-views relations support column gives idea about the considered approach, if it represents the relations that exist between the views, and what types of relations it represents.
- The stakeholder roles column defines the considered set of stakeholders for each approach.
- The Fixed/Not Fixed views column indicates either if the number of considered views in each approach is fixed or not.
- The hierarchy support column tells if each approach represents multiple levels of details/ abstraction levels/ hierarchy levels and gives a hint about its support if exists.

6 RELATED WORKS LIMITATIONS

Using viewpoints to describe software architecture benefits the development process in a number of ways the separation of concerns, communication among

Table 1: Existing works and their main characteristics.

	Inter-Views Relations Support	Stakeholder Roles	Fixed/Not Fixed Views	Hierarchy Support
(Delugach, 1990; Nuseibeh et al., 1994)	Represents dependency & overlapping	Development participants, End user	Not fixed	NA
(Sommerville and Sawyer, 1997)	Admits overlapping	Development participants, End user	Not fixed	NA
(Mullery, 1979)	Admits overlapping	Analyst, End user	Not fixed	Steps concept
(Nassar, 2003)	NA	End user	not fixed	NA
(Dijkman et al., 2008)	Represents refinement & overlapping	Development participants, End user	Not fixed	Single abstraction level per viewpoint
(Sowa and Zachman, 1992)	NA	Planner, Owner, Designer, Builder and Subcontractor	Fixed	NA
(Kruchten, 1995)	NA	Development participants	Fixed	NA
(Hilliard, 1999; ISO/IEC/IEEE, 2011; Rozanski and Woods, 2011)	Admits consistency relationships existence	Development participants, End user	Not fixed	NA
MoVAL	**Formal consistency relationships**	**Development participants, End user**	**Not fixed**	**Multiple abstraction levels per viewpoint**

stakeholder groups, complexity management and developer focus improvement. However, some limitations remain when using existing view-based approaches. We can summarize some of these limitations, which were solved in MoVAL, as follows:

- *Needs to move between different abstraction levels:* We assume that a software architect should define views at different levels of details. Thus, the software architect needs to think in terms of different abstraction levels and to be capable to move between them. Actually, in all the studied related works, an architect cannot specify a view or one of its representations in multiple abstraction levels, however in MoVAL he could identify for each view as many abstraction levels as he needs.
- *Lack of an Architectural Description Process:* In almost all the studied approaches, except in (Clements et al., 2002), it is unclear what the software architect has to do to define the suitable architectural description. This task is based mainly on his (her) experience and skills. So, MoVAL approach aims at defining an Architectural Description Process (ADP) which guides the software architect while defining the software architectural description. The ADP should be flexible, non-constraining and iterative. Actually, MoVAL's ADP is out of this paper's focus.
- *Views Inconsistency:* Using a number of Views to describe a system inevitably brings inconsistencies problems. So we need to achieve a cross-view consistency within an architectural description, which is not offered by the majority of the studied related works, but MoVAL has defined the links (section 4.3) which hold the needed coherence rules in order to solve those inconsistencies.

7 CONCLUSION

This paper presents an overview of several approaches using Views in different fields: Requirements Engi-

neering, Modeling, Implementation and Software Architecture. Limitations of existing approaches in Software Architecture field are emphasized. Also, this paper presents a preliminary approach for documenting intensive software systems architecture based on views and hierarchy levels. This approach complies with the IEEE standard 42010 and contributes with two major additions that are firstly, providing the software architect means to define views at different levels of detail and to move between them; then defining relationships that solve the inconsistencies between the architectural views.
Actually, we are preparing a complete methodology or development process in order to allow system architects to build a rough and coherent multi-views/multi-hierarchy software architecture.

REFERENCES

Anwar, A., Dkaki, T., Ebersold, S., Coulette, B., and Nassar, M. (2011). A formal approach to model composition applied to VUML. In *Engineering of Complex Computer Systems (ICECCS), 2011 16th IEEE International Conference on*, pages 188 – 197.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., and Stafford, J. (2002). A practical method for documenting software architectures.

Delugach, H. S. (1990). Using conceptual graphs to analyze multiple views of software requirements.

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

Finkelstein, A. and Fuks, H. (1989). Multiparty specification. In *ACM SIGSOFT Software Engineering Notes*, volume 14, pages 185 – 195.

Hilliard, R. (1999). Views and viewpoints in software systems architecture. In *First Working IFIP Conference on Software Architecture,WICSA*, pages 13 – 24.

ISO/IEC/IEEE (2011). Systems and software engineering – architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*.

Kruchten, P. (1995). The 4+ 1 view model of architecture. *Software, IEEE*, 12(6):42 – 50.

Ladeira, S. and Cagnin, M. I. (2007). Guidelines for business modeling elaboration based on views from domain information. In *10th Workshop on Requirements Engineering, Toronto-Canada*, pages 47 – 55.

Majumdar, D. and Swapan, B. (2010). Aspect Oriented Requirement Engineering: A Theme Based Vector Orientation Model. *Journal of Computer Science, InfoComp*.

May, N. (2005). A survey of software architecture viewpoint models. In *Proceedings of the Sixth Australasian Workshop on Software and System Architectures*, pages 13 – 24. Citeseer.

Medvidovic, N., Taylor, R. N., and Whitehead Jr, E. J. (1996). Formal modeling of software architectures at multiple levels of abstraction. *ejw*, 714:824 – 2776.

Mili, H. and al (1999). View programming : Towards a framework for decentralized development and execution of oo programs. In *Proc. of TOOLS USA' 99*, pages 211 – 221. Prentice Hall.

Mili, H., Sahraoui, H., Lounis, H., Mcheick, H., and Elkharraz, A. (2006). Concerned about separation. *Fundamental Approaches to Software Engineering*, pages 247 – 261.

Monperrus, M., Beugnard, A., and Champeau, J. (2009). A definition of "abstraction level" for metamodels. *7th IEEE Workshop on Model-Based Development for Computer Based Systems*.

Mullery, G. P. (1979). CORE-a method for controlled requirement specification. In *Proceedings of the 4th international conference on Software engineering*, pages 126 – 135.

Nassar, M. (2003). VUML: a viewpoint oriented UML extension. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, pages 373 – 376.

Nassar, M., Anwar, A., Ebersold, S., Elasri, B., Coulette, B., and Kriouile, A. (2009). Code generation in VUML profile: A model driven approach. In *Computer Systems and Applications, 2009. AICCSA 2009*, pages 412 – 419.

Nuseibeh, B., Kramer, J., and Finkelstein, A. (1994). A framework for expressing the relationships between multiple views in requirements specification. *Software Engineering, IEEE Transactions on*, 20(10):760 – 773.

Ossher, H. and al. (1995). Subject-oriented composition rules. In *OOPSLAS'95*, pages 235 – 250.

Perry, D. and Wolf, A. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40 – 52.

Regnell, B., Andersson, M., and Bergstrand, J. (1996). A hierarchical use case model with graphical representation. In *Engineering of Computer-Based Systems, 1996. Proceedings., IEEE Symposium and Workshop on*, pages 270 – 277.

Robinson, W. N. (1990). Negotiation behavior during requirement specification. In *Software Engineering, 1990. Proceedings., 12th International Conference on*, pages 268 – 276.

Rozanski, N. and Woods, E. (2011). *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley.

Sommerville, I. and Sawyer, P. (1997). Viewpoints: principles, problems and a practical approach to requirements engineering. *Annals of Software Engineering*, 3(1):101 – 130.

Sowa, J. and Zachman, J. (1992). Extending and formalizing the framework for information systems architecture. *IBM systems journal*, 31(3):590 – 616.