MoVAL, a new approach to software architecture and its comparison with existing views based approaches in software engineering.

AHMAD KHEIR[1,2]
HALA NAJA[2]
MOURAD OUSSALAH[1]

[1]University of Nantes
LINA - Laboratoire d'Informatique de Nantes Atlantique
[2]Lebanese University
EDST - AZM platform for research in biotechnology
[1](ahmad.elkheir,mourad.oussalah)@univ-nantes.fr
[2]hjazzar@ul.edu.lb

Abstract. Views and viewpoints are concepts usually adopted in an important number of works in software engineering in different domains, like in requirements specification, system modeling, system implementation, and mainly in software architectures.
This paper presents a survey about the use of viewpoints in these different domains, and leads a comparative synthesis between different approaches in order to induce their limitations. Also, it briefly presents the main characteristics of our views based approach MoVAL that solves two kinds of problems during view-based software development: the stakeholders' communication complexity and the lack of an architecture definition process guiding architect during architecture construction.

Keywords: Software architecture, Viewpoint, View, abstraction level.

(Received April 16th, 2014 / Accepted July 2nd, 2014)

1 Introduction

The viewpoint concept, also called view, takes various meanings across the field of computer science. In general, we are interested in views as soon as we model complex systems involving a large amount of data that require the cooperation of several experts from different fields of knowledge and different points of interest and addressing a wide range of users. As discussed in [17], a viewpoint links a designer, a universe of discourse (*i.e.* the system), and the goal that the designer is trying to achieve. The viewpoint enables a partial representation of the system to be modeled, highlighting one or more aspects of the latter and concealing others.

The first work on viewpoints falls within the field of knowledge representation in artificial intelligence. Here, we can mention the work of Minsky [15] followed by KRL languages [3], LOOPS languages [2], and ROME languages [5], which have all identified the need to bestow several representations to the same object.

In databases, viewpoint concept was introduced in 1975 for the first time in Ansi/Sparc's report under the name of external schema. An external schema represents the unique access point from which users can access data from the database. This is a part of the base, which interests a user or group of users and incorporates the concept of access rights. A viewpoint restricts the visibility of data and/or adapts their structure to the application requirements. It helps solve problems related to the configuration of user interface for data protection, for change in data organization without disrupting the existing organization as well as for query optimization.

In software engineering, the motivation of viewpoint is the separation of concerns. Thus, points of views are introduced as construction elements for the management of complexities of artifacts products (such as requirements specifications, design models and programs).

In section 2, we conduct a survey on views in four areas of software engineering, which are requirements specification, system modeling, system implementation, and software architectures. Then in section 3, we extend the review to software architecture as it is closely related with the purpose of our study, and we detail some software architecture approaches and present their limitations. In sections 4 and 5, a novel software architecture approach will be introduced and compared to the other approaches. Finally, section 6 concludes the paper.

2 Views in software engineering

In this survey we have conducted a study of views in three disciplines in software engineering, which are:

- Requirements specification: requirements specification for a software system is a description of the behavior of this system and the interactions users may have with it.
- System modeling: system modeling is a technique to express, visualize, analyze, and transform requirements of a system in software engineering.
- System implementation: system implementation is the process of defining how the information system should be built, ensuring that it suits well the needs of its users, and that it is operational.

2.1 Views in Requirements Specification

In 1979, Mullery developed a formal method for the specification and modeling of complex computer systems called CORE [16]. This method, proposes to break down the modeling process into several steps and, for each of them, to identify the viewpoints associated with them, so as to finally achieve a schematic representation of the system while taking into account all viewpoints of this system.

In [9] and [25] methods and tools are developed for the derivation of the computer systems specifications from a dialogue between different points of views. Delugash et al. [7] represented the needs associated with each viewpoint in a conceptual graph. Then, an integration of the different graphs into a single graph is carried out. The obtained graph specifies the requirements for the entire system. Nuseibeh et al. [20] represent the links and relationships that can exist between the points of views through a given platform. Finally, in 1997, Sommerville proposed his Preview approach [27], which gives architects of computer systems a method and tools to discover and analyze the requirements of different viewpoints.

2.2 Views in System Modeling

In terms of system modeling, in [17], the author defines five properties that an object-oriented model should have to make it a multiview object-oriented model. These properties are: (1) a range of one viewpoint at various levels, (2) multiple representation has a repository, (3) multiple representation is decentralized, (4) partial representations exchange information between them, and (5) multiple representations are consistent.

Also, Dijkman *et al.* [8] presented a platform in which the software architect can model different viewpoints of the system by determining a collection of basic concepts that are common to all viewpoints, and then by associating each viewpoint with a suitable level of abstraction and by defining the relationships that may exist between the viewpoints.

Still in modeling, UML notation embodies the broader concept of the viewpoint. In fact, in its current standard, the UML proposes 13 types of diagrams where each type bestows an implied viewpoint, from which a complex system will be approached, thus allowing the decomposition of complex model of a system into several less complex, more easily affordable and intelligible supplementary sub-models. In addition, UML proposes an extension mechanism that allows users to add or customize the predefined types of diagrams and thus to add points of views.

In VUML approach [1, 18], an extension of UML is proposed. It is based on the ability to coexist several class diagram for the same system, and then to proceed to an integration of these diagrams into a single one, consistent with the proposed VUML meta-model.

2.3 Views in System Implementation

In the history of system implementation and programming languages, the terms "decomposition" and "modularity" have always been keywords to reduce the complexity of programs. Thus, we obtain less complex, loosely coupled and easily reusable modules.

The separation of concerns aims to carry out a modularization of programs based on various concerns. The idea is to consider a system as a core with extensions. The core is the set of basic functional requirements for which the system is primarily designed and the extensions are additional secondary requirements, which add functionalities to the core. An extension is a concern that cross-cuts several basic functional requirements. The implementation of a concern gives rise to an aspect. Hence, the programming called "Aspect-Oriented Software Development" (AOSD) [12, 13]. Other paradigms exist based on the same principle; however, they diverge for several reasons, such as subject-oriented programming [8] or view oriented programming [13]. We do not extend the comparison between these different paradigms. The interested reader can refer to [12] for such a comparison.

3 Software architecture approaches and their limitations

Within software architecture, it is almost impossible to capture the architecture of a complex system in a single model that is understood by all stakeholders. By stakeholder, we include both the user of the future system and the manufacturer of the latter. A manufacturer is a person or a group of people that designs, develops, tests, deploys or maintains the future system.

In each of these activities, each stakeholder has its own needs, interests, requirements, and wishes that the system considers. Understanding the role and aspirations of each stakeholder is the role of the architect during the development of the system.

The recommended solution for an architectural description (AD) is to partition it into a number of separate and interdependent points of view, which collectively describe the functional and non-functional aspects (such as performance, robustness, availability, competition, and distribution) of the system. This solution is not new; in fact, it goes back to the work of Parnas in 1970 [22] and most recently, in 1990, to that of Perry and Wolf [23].

In addition numerous approaches and frameworks have been developed since that date, like the *"4+1" View Model* [11], *Siemens Model* [28], and most recently *Rozanski* and *Woods* approach [26], that targeted the decomposition of a software architecture to different predefined views in order to cover every aspect of a software system. Also, the *Reference Model of Open Distributed Processing* (*RM-ODP*) [24] is one of the important frameworks in this field that targeted the standardization of *ODP* (*Open Distributed Processing*) systems by supporting mainly the distribution, interworking, and portability aspects of these systems. *Views and Beyond* (*V&B*) [6] is an architecture description approach, in which authors have offered a detailed guide to software architects for architecture documentation using an organization of views and viewtypes. *Zachman* framework [29] have offered tools to classify a software architecture's artifacts in a matrix organization decomposing the architecture to six different views and defining for each of them six different focuses. Simultaneously, the *IEEE* standard have been defined for the first time in 2000 as a try to bring all the important existing works in this field, and create based on them a standard for multi-views software architectures. This standard has been revised in many other occasions, most recently in 2011 under the name *IEEE 42010* standard [10], to keep it synchronized with novel works in the field.

In what follows, we describe five approaches, which have provided satisfactory solutions to this problem.

3.1 "4+1" view model

The "4+1" View Model [11] proposed by P. Kruchten of the Rational Software Corp, is based on five main views, shown in Figure 1, which are as follows:

- The logical view covers mainly the functional requirements of the system, or in other words, what the system provides as services to users. The logical view shows the system objects and the relationships between them.

- The process view represents a non-functional aspect of the system requirements such as performance, availability, concurrence, distribution, and integrity. This view divides the system into a set of processes and represents the interactions that will take place between these processes.

- The development view focuses on the modular organization of the system. Thus, in this view, the system is broken down into several libraries or subsystems that can each be implemented by a developer or a small team of developers. This view covers the internal requirements of the implementation as reusability, compliance with standards and constraints, etc.

- The physical view takes mainly into account the non-functional system requirements such as availability, reliability, performance, and scalability. In the physical view, software elements, such as processes and objects, will be associated with different physical components such as processors and hard disks.

- Scenarios represent the integration of elements of the previous four views using a small set of scenarios that seem important.

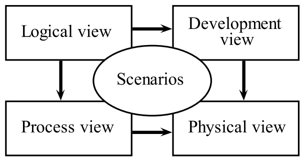


Figure 1: Kruchten's "4+1" model

The views offered in this model are not completely independent. In fact, matches can be seen between these views, for example as a match practice between a small class or set of classes, which will normally be represented as a module or a set of modules in the development view, or the process for the process view, which will be associated with the physical media of the physical view to be performed.

It should be noted that this model has been adopted in the iterative development process called unified process (UP) [4].

This model was extended in the *Rational Architecture Description Specification* (*Rational ADS*) approach [19] in order to have a better behavior with more complex systems. Thus, this approach added other views to the model and defined how those views are interdependent one to the other.

3.2 ISO/IEC/IEEE 42010

ISO/IEC/IEEE 42010 [10] was designed by the IEEE APG in 1996, approved by the IEEE-SA standards board and coded IEEE 1471-2000, then adopted by the ISO/IEC JTC1/SC7 and named ISO/IEC 42010:2007. This standard aims at formalizing the definition of software architecture and its main elements, and also in order to provide a common standard for the incorporation and embodiment of the efforts made in this field.

IEEE 42010 defined an architecture or AD as being the organization of a system, structured by a collection of components (i.e. units) and software links or relationships defined between these components.

According to the model proposed in this standard and presented in [10]:

- an AD addresses a broad array of stakeholders who have concerns. A concern may be covered by viewpoints:
- a viewpoint is a specification of the construction agreements of its own views. These agreements may be defined in the architecture itself or imported from an external entity called Model kind;
- a view is consistent to a viewpoint and consists of a set of models;
- an AD is made up of a set of views, so that each one complies with a viewpoint and is a set of models that describe the architecture.

3.3 *Rozanski* and *Woods* approach

Rozanski and *Woods* approach [26] complies with the *IEEE 42010* standard. The contribution of this approach is the definition of a fixed catalog of interdependent viewpoints. Those viewpoints are:

- Context: describing the relationships, dependencies, and interactions between the system and its environment.
- Functional: describing the functional elements of the system in execution time, their relationships, interfaces, and interactions.
- Information: describing how the system stores, manipulates, manage, and distribute the information or data.
- Concurrency: describing the concurrency structure of the system and highlighting the concurrent portions of some execution processes.
- Development: describing and communicating the architectural aspects of the system among all the stakeholders having concerns in the construction of the system.
- Deployment: describing the environment in which the system must be deployed, and the potential dependencies among its elements.
- Operational: describing how the system must be managed during execution time in its production environment.

Also, *Rozanski* and *Woods* have defined the dependencies between different viewpoints of the architecture, as illustrated in Figure 2.

Note that this approach was fortified with a global architectural description process that guides the software architect defining effectively his architecture.

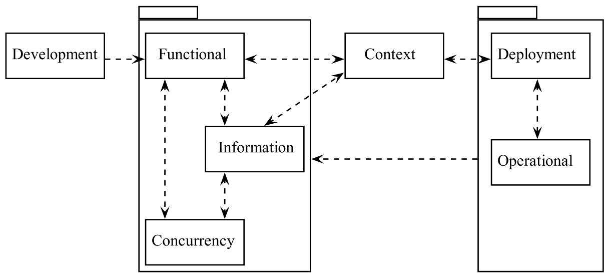


Figure 2: *Rozanski* and *Woods* approach viewpoints and their dependencies, extracted from [26].

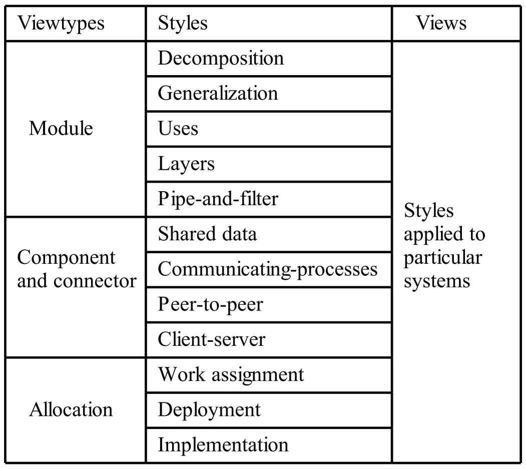
3.4 Views & Beyond approach

Within the SEI (Carnegie Mellon), Clements et al. developed the V&B approach [6] for documenting software architectures. This approach, as its name suggests, uses views to achieve a fundamental organization of software architectures. In fact, V&B approach is based on the principle that the documentation of a software architecture begins with the documentation of its relevant views and thereafter, documenting the information linking these views together.

Basically, V&B approach is for all viewpoints in the development process of the computer system, and aims to provide documentation of a software architecture that is decomposed into several views for different stakeholders meeting their requirements.

In this approach, a three-level hierarchy is defined as follows:

- The viewtypes: a viewtype represents a category of views and is for one or a set of stakeholders. There are three viewtypes: (1) module viewtype, (2) component and connector (C&C) viewtype, and (3) allocation viewtype.
- The styles: an architectural style also known as architectural pattern is a high-level pattern that assists to specify the basic structure of an application. Any style helps to achieve an overall property of the system, such as the adaptability of the user interface or distribution. Styles are grouped in viewtypes, which are considered as categories of styles. The list of styles defined for each viewtype is illustrated in Figure 3.
- The views: it represents collections of system elements and relationships that link them. Note that the architectural views are documented in a template defined by the designers of this approach. In fact, an architectural view is always consistent with a style.



Viewtypes	Styles	Views
Module	Decomposition	Styles applied to particular systems
	Generalization	
	Uses	
	Layers	
	Pipe-and-filter	
Component and connector	Shared data	
	Communicating-processes	
	Peer-to-peer	
	Client-server	
Allocation	Work assignment	
	Deployment	
	Implementation	

Figure 3: Viewtypes, styles and views in V&B approach

V&B approach offers a three-step guide for the selection of relevant views necessary to document the architecture of a system:

1. Produce a list of candidate views: this step is to construct a two-dimensional array (row-column) as the rows represent the stakeholders that the architect considers relevant to the current project and the columns denotes grouped views into viewtypes and the styles that can be applied. Then, fill in the boxes of the table with values describing the level of information required for each stakeholder and each style. The level of information can be: d for detailed information, s for some detail, and o for overview information.
2. Combine the views: this step aims to minimize the number of views obtained in step 1 by ignoring the views whose architectural value is covered in other views and by combining other views.
3. Sorting views: sorting the remaining views will be carried out in the order of priority of the documentation, depending on the specific details of the project.

The last phase of the documentation of a proposed architecture by the V&B approach is the documentation of inter-views information that are applied to multiple views. This phase has been proposed to link the views of the architecture together and give an overall picture facilitating the understanding of the stakeholders.

3.5 *Zachman* framework

Zachman framework [29] was designed basically to classify and to organize the models and artifacts of a software architecture based on two parameters, which are the viewpoint holding this model and its focus. Hence, a software architecture could be represented in Zachman framework via a matrix 6 $\times$ 6, where each row represent a viewpoint and each column represent a focus.

The six different views in Zachman framework, illus-

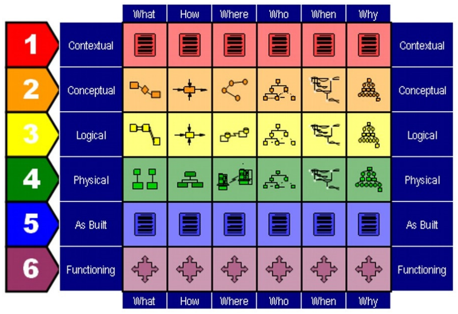


Figure 4: Different views and focuses defined in *Zachman* framework

trated in Figure 4, are:

- Scope (Contextual): where the system context, scope, and relationships with other systems are defined.
- Business model (Conceptual): in this view the architect must draw the vision of the system owner towards the business entities and processes of the system.
- Information system model (Logical): in this view, designers of the system must model the system requirements gathered in the previous views.
- Technology model (Physical): in this view, developers must show how to implement the design of the last view considering a specific technology.
- Detailed specification (As Built): represent the detailed implementation models.
- Actual system (Functioning).

Basically, in this framework each view is built by answering different questions representing different focuses for each view. Those questions and focuses are: data description (What), function description (How), Network description (Where), people description (Who), time description (When), and motivation description (Why).

3.6 Views contribution in Software Architecture

The introduction of viewpoints in software architecture contributed to the improvement of the process of describing architecture in several ways [26]:

- Separation of concerns: the separation of multiple aspects of a system using several different models during the design process, analysis, etc., helps a particular aspect while being able to focus on each step.
- Communication with stakeholders groups: communication between stakeholders groups with various concerns is a challenge for the architect. Approaches based on views provide an opportunity for stakeholders to converge relatively quickly toward the ADs that interest them and that respond to their concerns.
- Managing complexity: all aspects of a system, considered simultaneously in the same model, result into a complexity that a human being cannot handle. The thought of breaking down the model into models according to different views notably reduces the complexity.

3.7 Limitations of current approaches in software architecture

Within software architecture, approaches working on the view concept have effectively contributed in the software development process. In fact, they have led to the reduction of the complexity of an AD by combining all the information and treatments related to specific concerns; it is the complexity caused by the multitude of concerns. However, some limitations may be noted. Among these limitations, we note:

- The persistence of complexity within views: in fact, if the views have solved the problem caused by the complexity owing to the multitude of concerns raised above, an AD must be able to provide some solutions for another form of complexity: the intra-view complexity which is owing to the need to consider a hierarchical description at different levels. In fact, it is essential to take a hierarchical approach that reveals different levels of understanding in a view. In other words, the architect needs to decompose a view into several levels and needs to specify the links and types of links between these levels. Besides the fact that this description in multiple levels helps to reduce the complexity within a view, it assists to better meet the requirements of stakeholders which, depending on the circumstance, need to study a detailed or semi-detailed overall description of a view and navigate between these levels of description.

- The proposal of a process of AD considering different forms of complexity: for instance, the Kruchten [11] and Clements [6] approaches proposed a guide for the selection of viewpoints and views, but none of them have formally integrated levels of hierarchies, in conjunction with views.

- The problem of inconsistency between the views: no approach has brought a formal solution to the problem of inconsistency between the portions of descriptions raised from different views which are inevitably complementary and dependent.

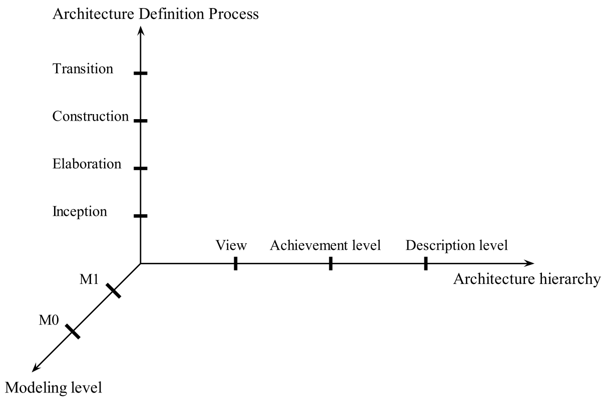


Figure 5: Triptych representing the intensions of the novel approach

4 MoVAL Novel approach for software architecture

In light of the study presented above, the motivations and goals of a novel approach have been fixed. It is a software architecture definition and documentation approach that aims to reduce different kinds of complexities that reside inside an architecture description. Thus, it addresses three problems during software development.

- Separation of concerns need: as it is claimed to be a view-based approach, it adopts views as a key concept to achieve the separation of concerns issue already identified in almost all approaches.

- Inner views complexity: usually a view is built using a huge amount of models. Not organizing those models would lead to a misunderstanding of the models purposes and links between them.
  Our approach proposes to use two refinement principles in order to organize models in an efficient and appropriate way. The two principles are: (1) Refinement to detail and (2) Refinement to achieve. Those principles are ensured via achievement and description levels.

- The lack of an architecture definition process for a well-organized views approach: this approach proposes an architecture definition process guiding the architect through the architecture definition task, integrating steps to define well-organized models inside views as well as means to specify links between different views.

The triptych of Figure 5 illustrates the intensions and aims of this approach. In this figure, the architecture hierarchy axis illustrates the entities that will compose the architecture, which are the views, achievement levels, then description levels.
The second axis, the Architecture Definition Process (ADP) axis, presents the four different phases of the process proposed in this approach, which are the inception phase, elaboration phase, construction phase, and transition phase.
The modeling level axis describes how this approach addresses the needs of stakeholders at different modeling levels. Like the software architects, analysts, and designers at the M1 modeling level, and the application architect (deployment engineers), and users at the M0 modeling level.

4.1 MoVAL Key concepts and characteristics

MoVAL (Model, View, and Abstraction Level based software architecture) is a software architecture that intends to offer to the software architect tools that simplify the tasks of construction and documentation of understandable software architectures that could be easily manipulated by different stakeholders.
MoVAL is a multi-viewpoints software architecture that consists on the description of a software architecture via multiple views, then hierarchize each view to different abstraction levels of two different types: (1) the achievement levels, and (2) the description levels.
In fact, this approach was designed primarily to fulfill the industrial needs. Thus, it complies with the most widespread software architecture standards, like the IEEE 42010 [10], MOF (Meta-Object Facility) [21], and MDA (Model Driven Architecture) [14].

4.1.1 Achievement level

An achievement level in MoVAL is a set of artifacts that defines an architecture at a specific phase of the development process. Thus, an achievement level must always provide further implementation details comparing with the other preceding levels. In other words, it answers better to the "How to" question. For instance, the answer of a "How to" question, given a UML use case diagram, would be sequence diagram. Indeed, the use case diagram refer to use cases names, as the use case sequences shows the realization of the latter use cases (i.e. their scenarios).

4.1.2 Description level

This type of abstraction levels in MoVAL allows the architect to provide multiple descriptions with different granularity description levels of the artifacts. A description level expands descriptions given in preceding levels. It answers better to "What more about" question. For instance, an answer to this question given a set of components names, would be to specify their interfaces.

4.1.3 Multi-levels architecture

Actually, a MoVAL architecture is a multi-views architecture. Thus, this architecture is decomposed into multiple views in order to reduce its structural complexity. Then, each view defines a set of achievement levels related each other by some formal architectural elements called "Links" that ensure the consistency between the view's achievement levels. From another side, each achievement level is described in multiple related description levels, leading to a significant reduction of the description complexity in the entire resulting architecture.

4.2 Case study

In order to clarify MoVAL concepts and confirm its contribution and utility in software engineering and complex systems development field, a case study will be considered.

This case study consists of an *eCommerce WebApp*, in which multiple stores would be registered and given virtual spaces to expose their products for sale.

In this context, three views could be considered:

- Physical view: which represent the view of the system deployer. Thus, it manipulates the hardware and software resources used for the deployment of such systems.
- Functional view: representing the functionalities that must be offered by the system.
- Site administrator view: representing the system as seen by the system administrator and considering his requirements.
- Store administrator view: representing the system as seen by the registered store administrator.

For the functional view for example, two distinct achievement levels could be considered. In the first achievement level, the functionalities offered by the system could be represented in one single description level, using "Boxes-and-lines" notation as illustrated in Figure 6.

Then, for the second achievement level, those func-

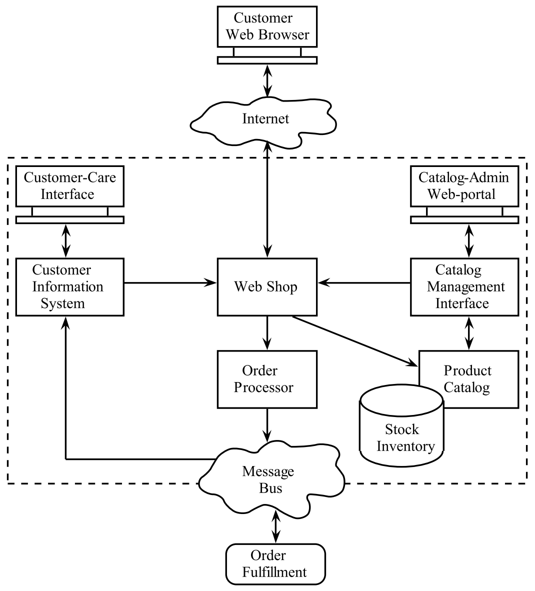


Figure 6: "Boxes-and-lines" model representing the first achievement level related to Functional view.

tionalities could be represented in a little more advanced technical way, using components structure. In this context, two distinct description levels could be considered for the achievement level of the functional view, representing in the first description level only the components as illustrated in Figure 7, and the components with their interfaces in the second description level.

4.3 MoVAL architecture definition process

MoVAL approach is endowed a specific architecture definition process, namely MoVAL-ADP, that complies with the Unified Process (UP) [4] and that was inspired from the process presented by Rozanski and Woods in [26].

This process is a set of activities and milestones that must be performed by the software architect in order to construct an appropriate software architecture that meets the stakeholders' requirements.

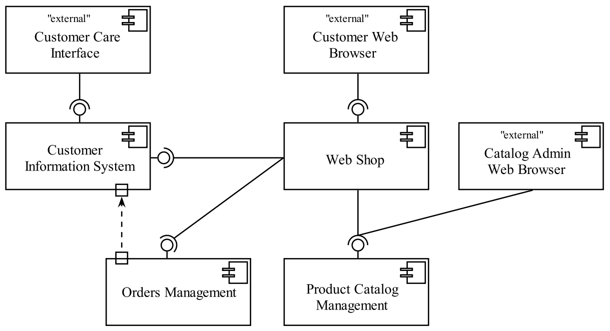


Figure 7: Components model representing the second achievement level related to Functional view.

Normally, as this process complies with the Unified Process, it defines four different phases, which are the inception, elaboration, construction, and transition phase.
In the inception phase, the architect must define the scope and context in which the target system shall be running, identify the main stakeholders, and capture the first-cut concerns and risks. Then, in the elaboration phase, the architect identifies the viewpoints of the architecture and creates the candidate architecture. Next, in the construction phase, he builds each view apart and defines its achievement and description levels and the links that relates them each other. Finally in the transition phase, the architect collects the implementation feedbacks and updates his architecture upon them. Due to space limitation, we will not detail MoVAL-ADP.

5 Comparison between MoVAL and other software architecture approaches

Following the previously discussed study, we make a comparative summary (see Table 1) between the five software architecture approaches and MoVAL approach based on the following seven criteria:

- Covered area:
  - the area covered by "4+1" View Model is the design of a software intensive system guided by scenarios and agreeing with incremental development processes such as RUP;
  - IEEE 42010 refers to the standardization of concepts and practices related to the description of a software-intensive system architectural design;
  - the area covered by Rozanski approach and MoVAL is the AD of a software intensive system walking through a specific ADP. MoVAL considers also hierarchizing the views of the architecture.
  - V&B approach is the documentation of an AD that is decomposed into several views for various stakeholders meeting their requirements.
  - Zachman framework covers also the design of software systems but while building a concrete software architecture.
- Approach focused on:
  - IEEE 42010, Rozanski approach, V&B approach, and MoVAL approach are based on the requirements of various stakeholders of the system being built;
  - the "4+1" View Model and Zachman framework are rather focused on the different phases of system development.
- Status/number of views: by status, we mean if the views are fixed and predefined by the approaches or if they can be created by the architect (by instantiating a meta-concept):
  - in "4+1" View Model, Rozanski approach, and Zachman framework: the views are fixed and their number is 5, 7, and 6 respectively. Notwithstanding this, all projects are not required to specify all the views, mainly in the "4+1" View Model and Rozanski approach but possibly a subset of them. The architect chooses this subset;
  - in V&B approach, combining existing views can create new views but the architect cannot introduce new styles and new categories of styles (viewtypes);
  - views in IEEE 42010 standard and MoVAL approach are not fixed neither predefined, but the architect define multiple views based on the domain and concerns of the system and associate them to different stakeholders.
- Categorization of views:
  - "4+1" View Model and Zachman framework have not defined a clear organizing principle for their views.
  - in IEEE 42010, Rozanski approach, and MoVAL approach, architectural views are categorized in viewpoints. A viewpoint is a specification of the construction agreements and the use of a view. A view must conform to a viewpoint;

  - in V&B approach, views are organized in viewtypes. A viewtype represents the structure of the system in terms of a set of elements and relationships between them according to agreements and notations defined in different styles;

- Architectural styles associated with views:
  - in "4+1" View Model, Kruchten noted the possibility of applying styles to different architectural views as, for instance, the object-oriented style for the logical view and the Pipes and Filters style for the process view, etc. However, he did not formalize the application of these styles;
  - IEEE 42010, Rozanski approach, and Zachman framework did not consider architectural styles for views.
  - in V&B approach, architectural styles are explicitly and formally associated with different viewtypes like the Pipes and Filters style associated with the C&C viewtype;
  - in MoVAL approach, architectural styles are not explicitly and formally associated with different views, but MoVAL-ADP specifies when and how those styles must be chosen.

- Integration of views:
  - within "4+1" View Model, the four main views (logical, development, process, and physical) are integrated through a fifth view, which is the "scenario" view, whereby the notation "4+1" view model emanates;
  - IEEE 42010 standard has not mentioned explicitly how different views must be integrated;
  - in Rozanski approach the dependencies map between different viewpoints of the architecture is provided in Figure 2;
  - in V&B approach, the view integration is considered in the second stage of the guide, where the designer combines the views which, according to him, seem close to one another and neglects the other views that focus on the details included in the other views.
  - Zachman framework has ordered its views in the main matrix of Figure 4, thus the integration of those views lies in this order for the reason that each view must offer every needed detail for the construction of the next view;
  - in MoVAL the integration of different views of the architecture is guaranteed in some architectural elements named links that offer to the architect tools to define and formalize the consistency between those views.

- Abstraction levels associated with views:
  - in "4+1" View Model, the IEEE 42010 standard, and Rozanski approach do not implement or mention any kind of abstraction;
  - to our knowledge, V&B approach is the only approach that (implicitly) introduced abstraction levels or information. These levels are considered in the first step of the process during which the designer must specify the level of information for each cell of the table built (detailed information, some information, and overview information);
  - Zachman framework has not mentioned explicitly the concept of abstraction but it defines six different focuses for each of its views. Those multiple focuses allow the architect to ignore some of the view's details in each focus. Thus, this feature could be considered similar in a way to the abstraction;
  - MoVAL appraoch has explicitly defined an hierarchy for each view of the architecture. This hierarchy contain two different type of abstraction, the achievement and description abstraction.

6 Conclusion

In this paper, a literature review is made on different approaches based on viewpoints and views arising from four fields related to software engineering, namely: requirements specification, system modeling, system implementation, and finally software architectures. In the field of software architecture, we identified the benefits and limitations of approaches, in which, in their proposals, the views, viewpoints, and abstraction levels have been incorporated.

Also, we have emphasized that the views and points of views are insufficient to develop the complex model of a software architecture and suggested the need for an hierarchical approach that reveals different levels of understanding in a view, thereby gradually controlling complexity.

Finally, we introduced our approach called *MoVAL*, which offers a description of a software architecture based on views, viewpoints, and abstraction levels. It helps to divide an AD into several views, each view

Table 1: A comparison between the three different approaches

	"4+1" View Model	ISO/IEC/IEEE 42010	Roazanski approach	V&B	Zachman framework	MoVAL
Area covered	Software design	Architectural description	Architectural description	Architectural documentation	Software design	Architectural description
Approach centered on	Different design phases	Stakeholders	Stakeholders	Stakeholders	Different design phases	Stakeholders
Status/number of views	Fixed/Five views	Variable/number defined by the architect	Fixed/Seven views	Variable/number defined by the architect	Fixed/Six views	Variable/number defined by the architect
Categorization of views	Informal	Viewpoint	Viewpoint	Viewtype	Informal	Viewpoint
Styles associated with views	Yes	–	–	Yes	–	Yes
Integration of views	Through the scenario view	–	Dependencies between the viewpoints	Through step 2 of the guide (combine views)	Views order in the matrix	Inter-views links
Abstraction associated with views	–	–	–	Informal (level of information)	Multiple focuses	Achievement and Description levels

containing models organized into two kinds of hierarchy: the first based on achievement levels; the second based on description levels.

As result, this approach helps for reducing the complexity of the models achieved and better meeting the expectations of various stakeholders, namely in the personalization of models, in their representations at gradual levels of complexities and thereafter, in a straightforward understanding of the models.

Currently, we are prototyping a *MoVAL*-specific tool that allow software development enterprises to adopt *MoVAL* approach and build hierarchical multi-views software architectures that complies with the *IEEE 42010* standard.

7 Acknowledgment

Authors would like to thank all who have contributed to the achievement of this preliminary and on-going study, especially the Lebanese National Council for Scientific Research (LNCSR), the Lebanese association for scientific research (LASeR), and CAPACITÉS association for funding this research project.

References

[1] Anwar, A., Dkaki, T., Ebersold, S., Coulette, B., and Nassar, M. 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, 2011.

[2] Bobrow, D. G. and Stefik, M. *The Loops Manual: Preliminary Vision*. Intelligent Systems Laboratory, Xerox Corporation, 1983.

[3] Bobrow, D. G. and Winograd, T. An overview of KRL, a knowledge representation language. *Cognitive Science*, 1(1):3–46, Jan. 1977.

[4] Booch, J. I., Grady and Rumbaugh, J. *The unified software development process*. Addison-Wesley, 1999.

[5] Carré, B., Dekker, L., and Geib, J.-M. Multiple and evolutive representation in the rome language, towards an integrated company information system. *TOOLS 1990*, 1990.

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

[7] Delugach, H. S. Using conceptual graphs to analyze multiple views of software requirements. 1990.

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

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

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

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

[12] Mcheick, H., Mili, H., Sadou, S., and El-Kharraz, A. A comparison of aspect oriented software development techniques for distributed applications. *IADIS press*, pages 324–333, 2006.

[13] Mili, H., Dargham, J., Mili, A., Cherkaoui, O., and Godin, R. View programming for decentralized development of OO programs. In *Technology of Object-Oriented Languages and Systems, 1999. TOOLS 30. Proceedings*, pages 210–221, 1999.

[14] Miller, J. and Mukerji, J. MDA guide version 1.0. 1. *Object Management Group*, 234:51, 2003.

[15] Minksy, M. A framework for representing knowledge. *The Psychology of Computer Vision, McGraw-Hill*, pages 211–277, 1975.

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

[17] Naja, H. La représentation multiple d'objets pour l'ingénierie. *Revue l'Objet: logiciel, bases de données, réseaux*, 4(2):173–191, 1998.

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

[19] Norris, D. Communicating complex architectures with UML and the rational ADS. In *Proceedings of the IBM Rational Software Development User Conference*, 2004.

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

[21] OMG. OMG meta object facility (MOF) core specification. In *OMG's industry-standard environment*, 2013.

[22] Parnas, D. L. Information distribution aspects of design methodology. 1971.

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

[24] Raymond, K. Reference model of open distributed processing (RM-ODP): introduction. In *Open Distributed Processing*, pages 3–14. Springer, 1995.

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

[26] Rozanski, N. and Woods, E. *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley, 2011.

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

[28] Soni, D., Nord, R. L., and Hofmeister, C. Software architecture in industrial applications. In *Software Engineering, 1995. ICSE 1995. 17th International Conference on*, pages 196–196. IEEE, 1995.

[29] Technology, M. Mdg technology for zachman framework user guide. Technical report, Zachman Entreprise, 2008.