Hierarchical Multi-Views Software Architecture

Ahmad Kheir, Mourad Oussalah
LINA Laboratory
Nantes University
Nantes, France
{Ahmad.Elkheir, Mourad.Oussalah}@univ-nantes.fr

Hala Naja
LaMA Laboratory
Azm Center For Research, Lebanese University
Tripoli, Lebanon
Hala.Naja@ul.edu.lb

***Abstract*—Software design and development hold so many inconsistencies when it comes to build composable and scalable structures. However, software architectures could be an efficient solution if considered with additional features like the composition of such architectures by linking different hierarchized views formally together. Thus, this paper presents a new contribution of a multi-views/multi-hierarchy software architecture that is consistent with the ISO/IEC/IEEE 42010 standard, and that presents a way for defining formally the consistencies between its different views and hierarchy levels.**

Keywords-Software architecture; Views; Hierarchy levels; Consistency

I. INTRODUCTION

Software architectures have contributed effectively in complex and distributed software systems development. Normally, there are two principles, which have made the software architectures' contribution obvious and indispensable. First, it allows the architect to model the structure and the behavior of the system simultaneously. Second, it offers the architect the base to build multi-hierarchy based models.

In fact, coherent and well organized software architecture would enhance some crucial system properties like the reliability, consistency, and scalability. However, the lack of such architectures may limit those systems' adaptability, evolution, and consequently their life cycle, due to the incapability of modifying or expanding the stakeholders' requirements.

This paper presents a Model, View and Abstraction Level based software architecture (MoVAL), a multi-views and multi-hierarchy software architecture, which complies with the IEEE standard 42010-2011 [1] and is based on the construction of multi-views models having for each of their views a hierarchy of levels.

Actually, the concept of viewpoint was present in many fields of software engineering domain. Indeed, it was introduced in requirements engineering by A. Finkelstein [4] in 1989 opening the way for other valuable works in this field like in [5] and in [6]. Also, the viewpoint concept was existing in software modeling, implicitly in some cases like in the unified modeling language (UML), where each diagram type has an implicit viewpoint, and explicitly in other studies like in the View-based UML extension (VUML) [7], where an explicit representation of different viewpoints in a single multi-views class diagram is proposed. Also, the software implementation field recognized the utility of viewpoint concept. Indeed, different development paradigms encapsulate the viewpoint concept, like the aspect oriented [8], subject oriented development paradigms [9] and the view-based programming technique [10], which define explicitly different views in a single model. In addition, most of the related works done in the field of software architecture like the *4+1 View Model* [2] and the *Views and Beyond* [3] approaches, have defined multi-views software architecture. However they did not provided any type of hierarchy for their views in order to reduce their complexities, nor they defined formally some consistency rules between different views of an architecture in order to conserve the robustness of that architecture and its ability to evolve while the stakeholders' requirements evolve. A complete survey on related works and a fruitful analysis of their limitations was presented in a previous study [11], but we can summarize those limitations in three main points: the views inconsistencies, the need to move between different abstraction levels, and the lack of a complete architectural description process.

In light of the related works study, MoVAL's motivations and goals were made clear. Actually, there are two main goals that were intended in this approach. The first goal is to propose a multi-views software architecture defining for each view a multi-levels hierarchy aiming to minimize software systems complexity per modeling entity. The second goal addressed in this approach, is to define formally the relationships that may exist between different views of a model, and also between different hierarchy levels inside a given view.

This paper is organized as follows: Section II presents in details our contribution. Then, the proposed approach is illustrated by a case study in Section III. Finally, Section IV concludes the paper.

II. MOVAL

In MoVAL, a model is conceptualized via a matrix as illustrated in Figure 1.

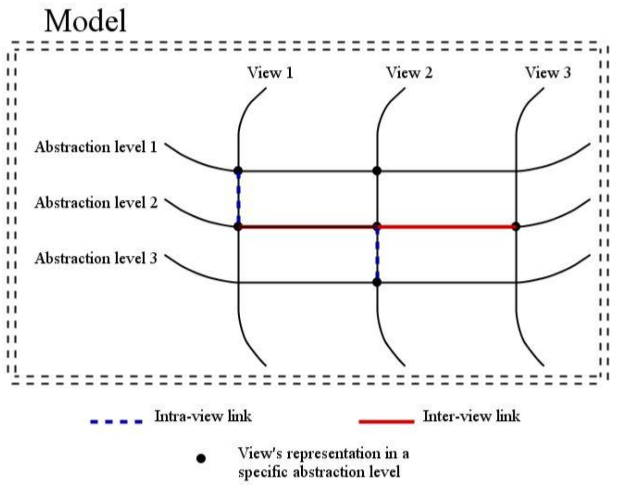

Figure 1. Conceptual Matrix of a MoVAL model.

The columns of the matrix represent the views of the model, while the lines represent its abstraction levels, which are the first level of the views' hierarchy detailed further in this paper. Hence, the lines and columns of the matrix illustrate two distinct structuring types defined in MoVAL. The columns illustrate the vertical structuring referring to different views of the same model, and the lines illustrate the horizontal structuring referring to the hierarchy levels defined in the model and associated to its views.

Note that model's matrix, in some trivial cases where the architect decides to create only one view for the model, and decide to represent this unique view in a single abstraction level, could be reduced to a single element.

A. *Model View*

A model view in MoVAL, or simply a view, is a representation of this model considering, from one side, a set of the development process' aspects, and from another side certain problems associated to a specific category of stakeholders or a group of categories of stakeholders. Those development aspects and problems are grouped in a separate entity, named viewpoints. In general, every stakeholder needs to express his interests via some appropriate semantics, syntax, and tools, called formalisms. For example, a database administrator needs to use the entity-relationship diagrams (ERDs) and the appropriate tools in order to model his database in a given phase. Thus, a viewpoint also defines the formalisms that shall be used afterwards to model the inherent views. Hence, each view must be associated to a specific viewpoint, which should be either predefined like the physical, structural, and behavioral viewpoints, or customized based on the application domain like the thermic view in an automobile construction system.

B. *View's Hiearachy Level*

MoVAL approach has defined a hierarchy of levels for each view, in order to describe it formally and appropriately in each step of the development process.

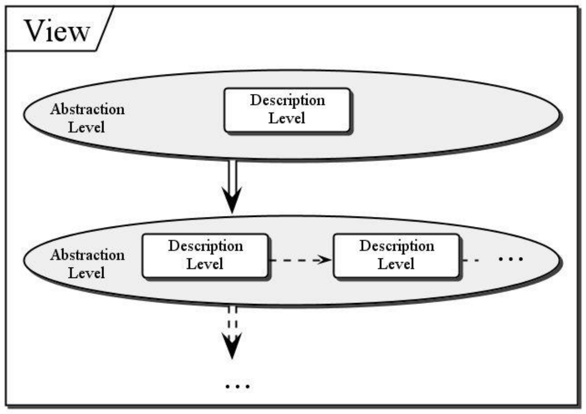

Figure 2. Views and hierarchy levels.

Figure 2 represents this hierarchy, which consists of two types of levels, the abstraction levels, which are represented in the figure via ovals. Also, under each abstraction level several description levels are represented via correlated rectangles.

1) *Abstraction Level*

An abstraction level is a representation of a view considered at a specific stage of the system lifecycle. Eventually, several abstraction levels could be considered on the same view, and then linked together by higher/lower relationships. In fact, for the same view, an abstraction level AL1 is higher than another abstraction level AL2 (*resp.* AL2 is lower than AL1) if AL1 defines relevant requirements in a given stage of the system lifecycle leaving out some other requirements and relegating them to AL2 in a more advanced stage.

For a given view, an abstraction level must use appropriate formalisms that are implied by the associated viewpoint.

In general, a view could have more than one abstraction level having the same inherent requirements as long as they have different formalisms. Actually, in this case the transition from one abstraction level of a view to another abstraction level in the same view conserving the same inherent requirements and changing the formalism, could indicate the transition from a stage of the software lifecycle to another more advanced stage.

Note that it is not mandatory to have always an isomorphism between different views of a model, by the fact that it is not mandatory to have each abstraction level associated to all the views of the model, as illustrated in Figure 1.

2) *Description Level*

The second type of hierarchy levels of a view is the description level. This type of hierarchy levels allows the architect to describe the same abstraction level of a specific view and the same inherent requirements while providing multiple descriptions having different granularity levels.

Here also, the description levels of the same abstraction level are linked together by higher/lower relationships. So, a description level DL1 is higher than another description level DL2 (*resp.* DL2 is lower than DL1) if DL1 lies on the same requirements as DL2 but adds more details in order to make easier the understanding of DL2's requirements. In other words, DL1 is at a higher granularity level than DL2.

Actually, the difference between this type of hierarchy levels and the abstraction levels, resides in the fact that a lower abstraction level allows the architect to go straightforward into more advanced stages of the system lifecycle relatively to the higher abstraction level, in general, by providing more requirements. However, a lower description level does not allow the architect to provide additional requirements of a specific view, but it allows him to describe more clearly its previous description level by providing more description details.

C. *Link*

The links are structural elements defined in MoVAL in order to express formally the relations between different hierarchy levels and conserve model's consistency. Those links are grouped in four categories:

- **Inter-views link**, defining the relation among a couple of distinct hierarchy levels belonging to two different views.
- **Inter-levels link**, defining a similar relation to that defined by the inter-views link, except that the hierarchy levels here belong to the same view.
- **Intra-level link**, defining an internal relation between elements of the same hierarchy level.
- **User links**; this category of links is a special category. A user link always inherits from one of the three previous categories, then defines some additional structural or semantic properties and attributes (see the case study in Section IV). Actually, the purpose of this category of links was to enhance the modularity and reusability of software architecture's structural elements.

In order to formalize the links, MoVAL has attributed four main properties to define them:

- **Source**: based on the semantic role of a link, its source could be either an abstraction or a description level of a view.
- **Destination**: similarly, the type of the destination of a link depends on its semantic role. Note that always the source and destination of a link must have the same type.
- **Semantic role**: the semantic role of a link defines the nature or the purpose behind the relation between the source and destination hierarchy levels. It is firmly related to the category of the link and the type of its source and destination hierarchy levels. Hence, MoVAL has defined three main semantic roles:
  - **Connection**, specifying some consistency rules between elements of the same hierarchy levels. Note that this semantic role could be used only for intra-level links.
  - **Composition**, specifying the composition of elements of the source level in the destination level, which is in this case the lower level. This role could be used in case of inter-levels or inter-views links.
  - **Expansion**, representing the description of elements of the source level in the destination level, which is in this case the lower level, respecting the abstraction levels of the source and destination. Actually, this semantic role is dedicated for the representation of relations between abstraction levels only and could be used in both cases of inter-levels or inter-views links.

  Normally, composition and expansion roles are adequate when the architect adopts a Top-Down development strategy. However, when the Bottom-Up strategy is adopted, composition and expansion could be replaced by other roles having inverse semantics, which are respectively the aggregation and compression semantic roles.
- Semantic link, which includes a set of semantic attributes aiming to implement the desired semantics, chosen in advance by architect via the semantic role:
  - **Dependence**, declaring that the destination hierarchy level depends for its existence on the source hierarchy level.
  - **Predominance**, which declares semantics symmetric to those declared by the dependence attribute.
  - **Coherence**, specifying that some consistency rules should be considered and respected in the destination hierarchy level based on the source level parameters, in order to conserve the coherence of the model. Those consistency rules could be expressed via a given constraint language like OCL.

D. *MoVAL Meta-Model*

MoVAL meta-model is consistent with the ISO/IEC/IEEE 42010 standard. Thus, some elements have kept their definitions presented in the IEEE standard, like the definition of a system, architecture, architectural description, stakeholder, viewpoint, view, and concern. However, some other elements were given new definitions like the model, and others have been introduced like the abstraction and description level, formalism, and link. Figure 3 presents the proposed meta-model.

A *System*, as it was defined in the IEEE standard, is not limited to individual applications but it encompasses them to cover also the subsystems, systems of systems and all kind of software interests' aggregations. A system always has different categories of *Stakeholders*, which are the participants in every phase of his life cycle. They could be individuals, teams or even organizations interested in this system, like the system architects, developers, analysts, experts contributing in the system development, users, etc.

Each of those stakeholders focuses on a specific part of the system requirements saturating his interests. Hence, those interests of different stakeholders are defined as different sets of *Concerns* overlapping in certain cases and contradicting in other cases.

Simultaneously, a system is associated to an *Architecture*, documented and described via an *Architectural*

Description (AD). An AD is composed of a set of *Views* governed by a set of *Viewpoints* specifying and grouping the inherent concerns and formalisms that should be used for the development of the views. Those views are represented in a hierarchy of *Abstraction* and *Description Levels*.

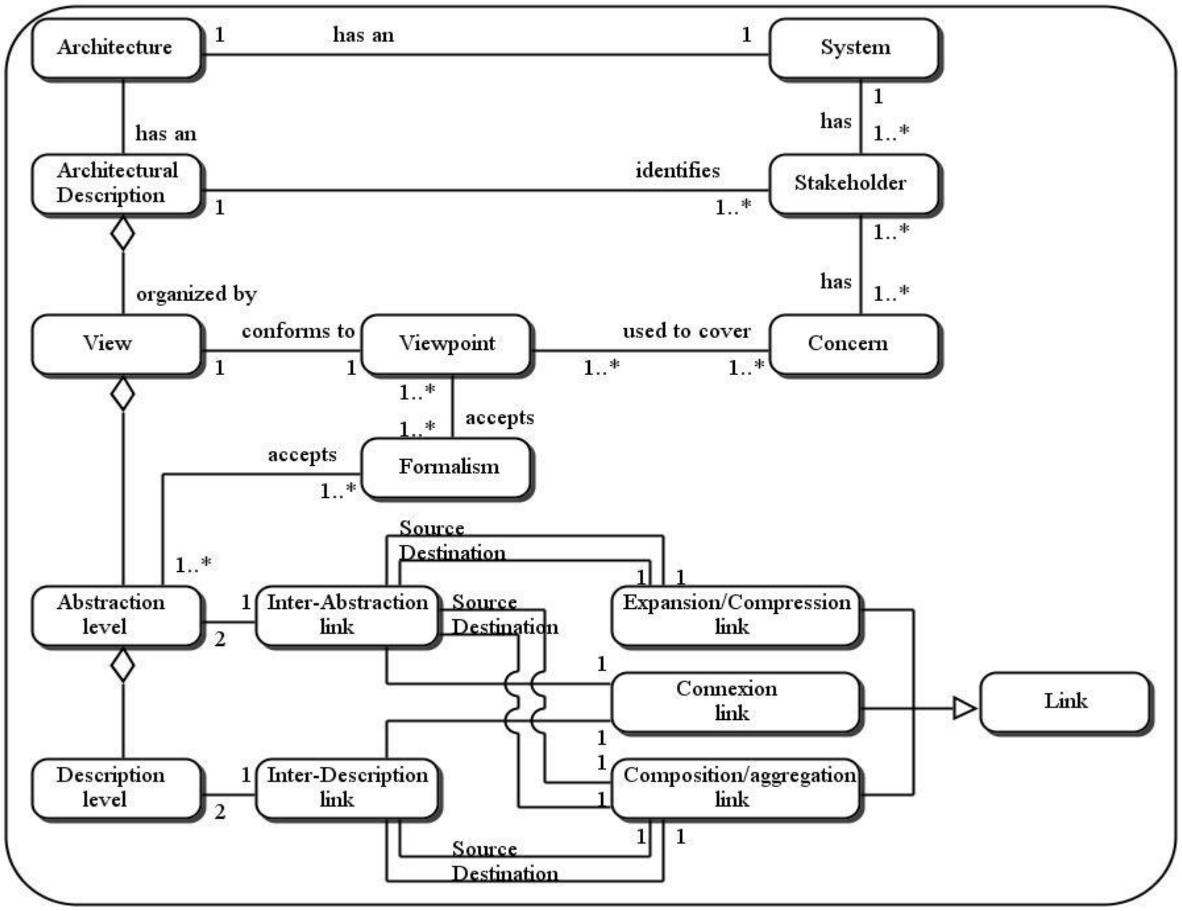

Figure 3. Conceptual model of MoVAL

Also, each viewpoint and each abstraction level of a model offers a set of Formalisms that could be used afterward to model the associated view at each of its abstraction levels. Those formalisms define actually the lexical and syntax elements that could be used.

III. 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 represented in this section.

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

In this context, only three viewpoints are considered (due to space limitation issue):

- ✓ **Physical** viewpoint, which represents the view of the system deployer. Thus, it manipulates the hardware and software resources used for the deployment of such systems. Actually, this viewpoint is predefined in MoVAL and considered associated to a single formalism, which is the deployment diagram of UML. The associated view could be represented in a hierarchy of one abstraction level and one description level mentioned respectively in figures 4 and 5.

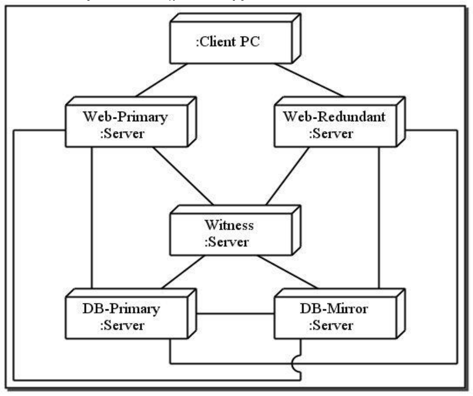

Figure 4. Physical view abstraction level.

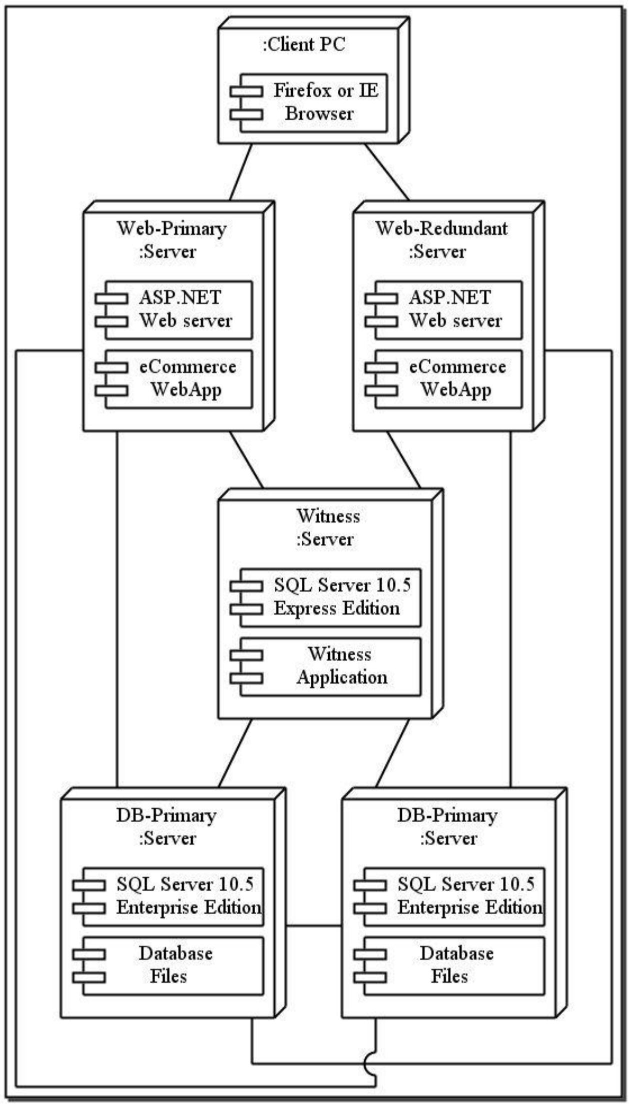

Figure 5. Physical view description level

- ✓ Site administrator viewpoint, representing the system as seen by the system administrator and considering his requirements. Three formalisms could be associated to this viewpoint, which are the use case, sequence, and class diagrams of UML. In addition, the associated view could be defined in two abstraction levels illustrated in figures 6 and 7, respectively.

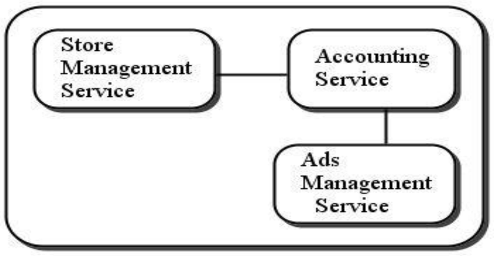

Figure 6. First abstraction level of the Site admin view.

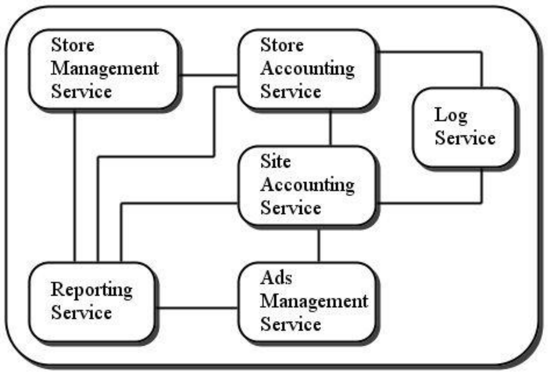

Figure 7. Second abstraction level of the Site admin view.

- ✓ Store administrator viewpoint, representing the system as seen by the registered store administrator. This viewpoint will be associated to the same formalisms associated to the previous viewpoint, also the associated view will be defined in two abstraction levels illustrated in figures 8 and 9.

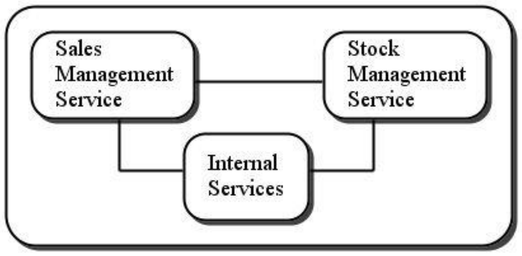

Figure 8. First abstraction level of the Store admin view.

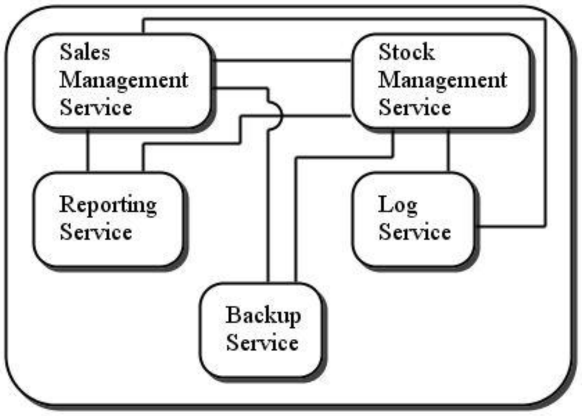

Figure 9. Second abstraction level of the Store admin view.

In general, in order to improve the models' consistency, the system architect must create different links between different views and hierarchy levels of this model. For this reason, the abstraction levels of the *Site Administrator* view could be associated to the abstraction levels of the *Store*

Administrator view, as they share the same level of details. Thus, for the remaining of this section, the higher and lower abstraction levels, associated to this couple of views, will be referred by the *First Functional Level* and the *Second Functional Level*, respectively.

Now, three links, among others, could be derived for this case study:

- ✓ Inter-levels link having the *First Functional Level* of the *Site Administrator* view as source hierarchy level, and the *Second Functional Level* of the same view as destination. This link is a composition link expressing in his coherence semantic attribute the composition of the *Accounting Service* in the source by the *Site Accounting Service* and the *Store Accounting Service* in the destination.
- ✓ Inter-levels link having the *First Functional Level* of the *Site Administrator* view as source and the *Second Functional Level* of the same view as destination. This link is an expansion link expressing in his coherence semantic attribute the expansion of the *Internal Services* of the source level to the *Log Service* and the *Backup Service* in the destination.
- ✓ User link, named *Reuse Link*, created by the architect as an inter-views link defining the reusability of a component of the source level in the destination level. Hence, a Reuse link could be defined having the *Second Functional Level* of the *Site Administrator* view as source and the *Second Functional Level* of the *Store Administrator* view as destination. This link expresses the reusability of the *Reporting Service* in both of the source and destination levels.

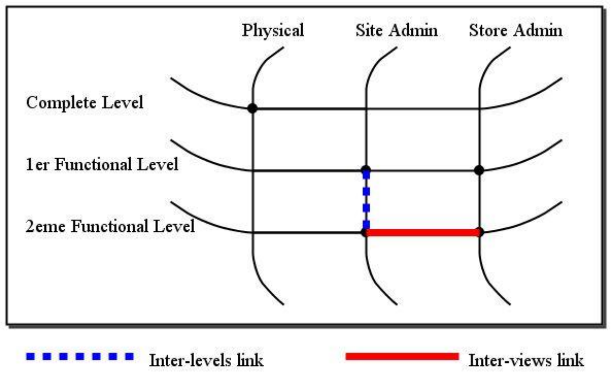

Figure 10. Conceptual matrix of the eCommerce model.

Figure 10 represents the conceptual matrix of the eCommerce case study.

IV. CONCLUSION

This paper has presented a new contribution of multi-views and multi-hierarchy software architecture, named MoVAL, defining and modeling independently, for each stakeholder, its inherent concerns in a separate multi-levels view and providing the necessary definitions to combine and link all those views and hierarchy levels in order to guaranty a complete consistency between different parts of the resulting architecture.

In fact, MoVAL has given every stakeholder the space to model his interests and the tools to represent the possible interferences that may exist with other interests of other stakeholders, what should decrease significantly the number of unexpected executions or the number of bugs of the system, and increase consequently the system's reliability.

From another side, MoVAL has given the software architect the tools to link different semantically related views or abstraction levels via the architectural links, what would enhance the model coherence because of the representation of every constraint that may exist between different views or abstraction levels. Simultaneously, this organization and coherence make the addition of other user requirements much simpler, and consequently increase model's scalability.

Actually, MoVAL is in the prototyping phase. A specific framework encapsulating the all the tools and features needed to apply MoVAL's concepts will be implemented and validated.

ACKNOWLEDGMENT

This material is based upon work supported by the Lebanese association for scientific research (LASeR), the Lebanese council for scientific research (CNRS), the French association CAPACITES, the University of Nantes, and finally the Lebanese University.

REFERENCES

[1] ISO/IEC/IEEE, “Systems and software engineering -- Architecture description,” in ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000), 2011.

[2] P. B. Kruchten, “The 4+ 1 view model of architecture,” IEEE Software, vol. 12, no. 6, 1995, pp. 42–50.

[3] P. C. Clements et al., “A practical method for documenting software architectures,” Research showcase, Carnegie Mellon University, 2002.

[4] A. Finkelstein and H. Fuks, “Multiparty specification,” in ACM SIGSOFT Software Engineering Notes, vol. 14, 1989, pp. 185–195.

[5] B. Nuseibeh, J. Kramer, and A. Finkelstein, “A framework for expressing the relationships between multiple views in requirements specification,” IEEE Transactions on Software Engineering, vol. 20, no. 10, 1994, pp. 760–773.

[6] I. Sommerville and P. Sawyer, “Viewpoints: principles, problems and a practical approach to requirements engineering,” Annals of Software Engineering, vol. 3, no. 1, 1997, pp. 101–130.

[7] M. Nassar et al., “VUML: a Viewpoint oriented UML Extension,” Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE'03), IEEE Computer Society, 2003, pp. 373–376.

[8] D. Majumdar and S. Bhattacharya, “Aspect Oriented Requirements Engineering: A Theme Based Vector-Orientation Model,” Infocomp J. Comput. Sci., vol. 9, no. 1, 2010, pp. 61–69.

[9] H. Ossher, M. Kaplan, W. Harrison, A. Katz, and V. Kruskal, “Subject-oriented composition rules,” in ACM SIGPLAN Notices, vol. 30, no. 10, 1995, pp. 235–250.

[10] H. Mili, J. Dargham, A. Mili, O. Cherkaoui, and R. Godin, “View programming for decentralized development of OO programs”, in Technology of Object-Oriented Languages and Systems (TOOLS 30), 1999, pp. 210–221.

[11] A. Kheir, H. Naja, M. Oussalah, and K. Tout, “Overview of an Approach Describing Multi-Views/Multi-Abstraction Levels Software Architecture,” Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2013), France, 2013, pp. 132–140.