Improving the IEEE std 1471-2000 for Communication among Stakeholders and Early Design Decisions

Luis Felipe Fernández Martínez, Cuauhtémoc Lemus Olalde, Miguel Serrano Vargas
(luisf, clemola, masv)@cimat.mx
Center for Research in Mathematics, A. C. (CIMAT)
Guanajuato, Gto.
México

Abstract

There are, at least, three reasons why software architecture is important: a) communication among stakeholders, b) early design decisions, and c) transferable abstraction of a system. The IEEE Recommended Practice for Architectural Description of Software-Intensive Systems introduces and integrates stakeholders, concerns, viewpoints, views, and architectural models facilitating the expression, communication, evaluation, and comparison of architectures in a consistent manner. However, the standard does not specify a delivery format for architectural description. In addition, it is difficult to know if an architecture is within the principles of design imposed by a specific concern. A similar effort, to describe software architectures is the creation and improvement of special-purpose languages, known as architecture description languages (ADLs). However, ADLs have the disadvantage of not providing adequate support for separating several kinds of concerns across different viewpoints. In order to alleviate these issues, our paper proposes an enhancement to the conceptual model introduced in the standard. Our enhanced model, improves two of the reasons mentioned: a) communication among stakeholders and b) early design decisions.

Keywords: Software architecture, software design and development, IEEE std 1471-2000.

1. Introduction

The attention given to issues of software architecture is increasing in both the software engineering research community and standardization organizations. Fundamentally, there are, at least, three reasons why software architecture is important [2]: a) communication among stakeholders, b) early design decisions, and c) transferable abstraction of a system. In this sense the IEEE std 1471-2000 effort [1], recommends architectural description practices for software intensive systems, seeking a common frame of reference to codify common elements between different architectural initiatives. The standard makes a clear distinction between the architecture of a software system and its description. The IEEE std 1471-2000 presents one consistent set of definitions targeting architectural descriptions. It involves stakeholders, concerns, viewpoint, view, and architectural models. The standard introduces a conceptual model of an architectural description that encompasses these concepts. Figure 1, shows how these concepts are related. However, the standard does not specify a delivery format for architectural description; in fact, it does not propose a way to express the richness of the model. There is another aspect that the IEEE 1471 does not take into account: How is it possible to know if an architecture is within the principles of design imposed by a specific concern? Hence, and assessment of the architecture must be performed.

A similar effort comes from the research community that has focused on the creation and improvement of special-purpose languages to describe software architecture, known as architecture description languages (ADLs) [8], [9] and [10]. Due to their formal nature, ADLs may be difficult to understand and use. However, ADLs also have the disadvantage of not providing adequate support for separating several kinds of concerns across different viewpoints. In addition, ADLs do not address the clear difference between software architecture and its representations, as does the IEEE 1471 [11].

In many occasions when practitioners or academics address an architecture, the concepts stated in IEEE 1471 (stakeholders, concerns, viewpoint, view, and architectural models), are intuitively and implicitly expressed. It is not clear however, if the design has followed specific principles of design and under what concerns the architecture was developed, as it is proposed by the standard.

In order to help improve this situation, our paper proposes an enhancement of the conceptual model introduced in the standard. As a first step, the model is seen from a perspective of class diagram, and not just as entities related among them. Second, we added attributes in some classes of the model, associating a metrics class to architectural models, and their relationship to concerns. Then a format is introduced using OCL (Object Constrain Language). Finally, we propose a semi-formal notation for architectural model to express in a condensed form most

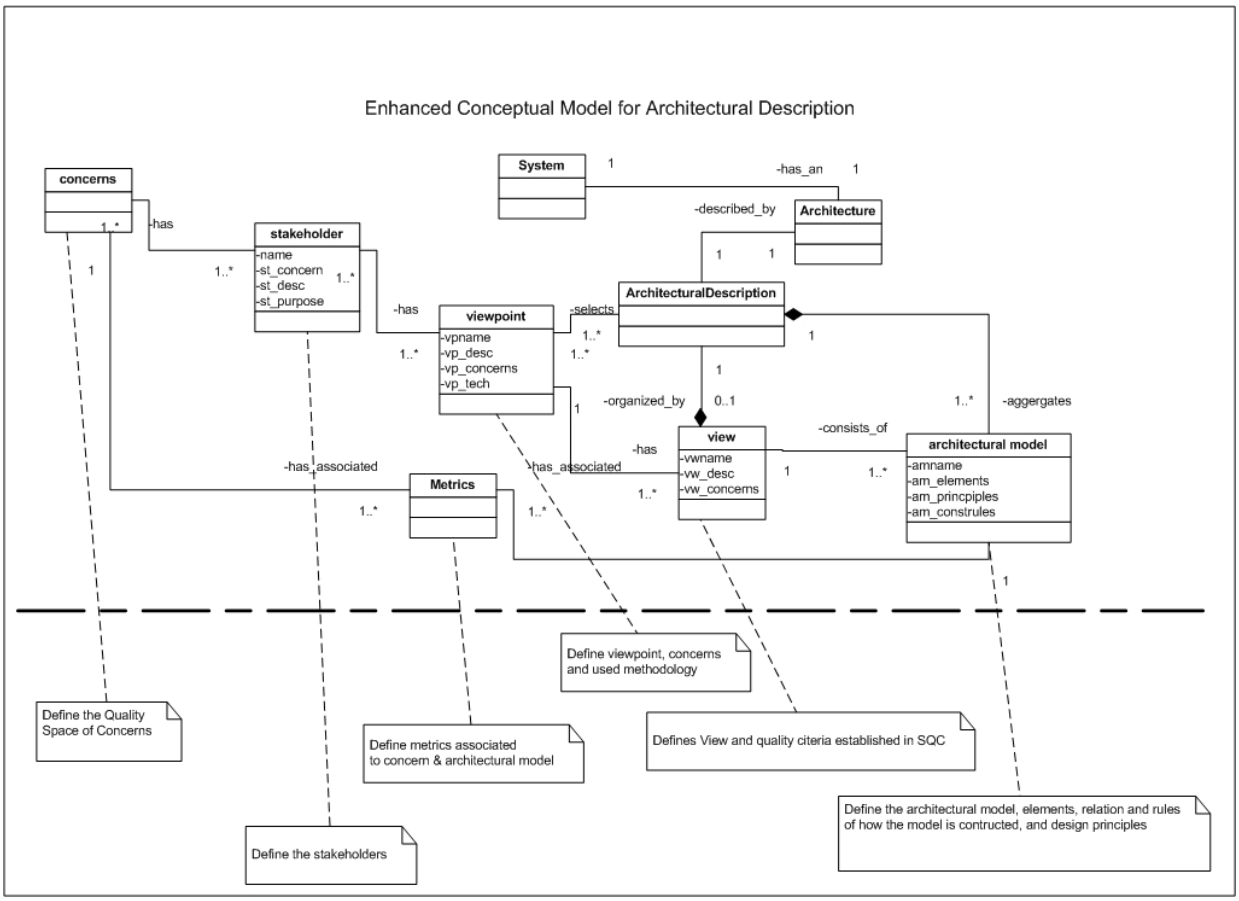

Figure 1: Conceptual Model of Architectural Description (Enhanced)

of the information. The paper is organized as follows. Section 2 offers an overview of IEEE 1471-2000 standard. Section 3 exposes a brief discussion about software architecture, and introduces a notation for architectural model. Section 4 introduces the enhancement to the conceptual model proposed in the standard. Also in section 4, a brief discussion of the approach and a simple example is presented. Finally, section 5 summarizes the paper and discusses future work.

2. IEEE std 1471-2000 overview

The IEEE std 1471-2000 is a recommended practice that addresses the activities of the creation, analysis, and sustainment for architectures of software-intensive systems, and the representation of such architectures in terms of architectural descriptions. The purpose of this recommended practice is to facilitate the expression and communication of architectures among stakeholders.

2.1 The conceptual model of IEEE std 1471-2000

This standard introduces a conceptual model, or frame of reference, for architectural descriptions. The model establishes terms and concepts pertaining to the content and use of architectural descriptions. For our work, the model is seen from a perspective of class diagram, and not just as entities related among them. As depicted in Figure 1, every system has an architecture. An architecture is expressed by an architectural description. According to Figure 1, a system has one or more stakeholders. Each stakeholder typically has concerns relative to that system. *Concerns* are those interests, which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders. Concerns include system considerations such as performance, reliability, security, distribution, and evolvability. An architectural description is organized into one or more constituents called (architectural) *views.* Each view addresses one or more concerns of the system's stakeholders.

A viewpoint establishes the conventions by which a view is created, depicted and analyzed. In this way, a view conforms to a viewpoint. The viewpoint determines the languages (including notations, model, or product types) to be used to describe the view, and any associated modeling methods or analysis techniques to be applied to these representations of the view.

An architectural description selects one or more viewpoints. The selection of viewpoints typically is based on considering the stakeholders to whom the architectural description is addressed and their concerns. A view may consist of one or more *architectural models*. Each architectural model is developed using the methods established by its associated architectural viewpoint. An architectural model may participate in more than one view. The IEEE 1471 does not address architectural models definition in a clear manner. In the next section, we clarify this concept.

3. Architecture of a software system

Although software systems have had architectures since the early days of computers, it has been recognized recently its relevance to specify, analyze and design software architectures. Software architecture is concerned with understanding and describing complex software-intensive systems at different levels of abstraction. Software architecture continues to present formidable challenges and difficulties in its design, construction, deployment, and evolution. Recent attempts to address these difficulties have focused on the earliest period of design decision-making and evaluation, increasingly referred to as the *architectural level* of system development. The concepts *architectural level* and *architecture*, if imprecisely, are widely used. Their use reflects acceptance of an architectural metaphor in the analysis and development of software systems. A key premise of this metaphor is that important decisions may be made early in system development in a manner similar to the early decision-making found in the development of civil architecture projects. This early decision-making is related to the stakeholder and concern concepts introduced in the IEEE 1471.

3.1 Software architecture in the IEEE perspective

At the essence of all the discussion about architectural metaphor in the analysis and development of software systems is a focus on reasoning about the structural issues of a system. In the same way that a building exhibits many structures, a software system has many architectural structures that are views capturing different aspects of the system, but as a whole describe the overall architecture of the system. The conceptual model introduced in IEEE std 1471-2000 [1] covers this issue in a good way. The IEEE 1471 makes a clear distinction between the architecture and the architecture description of a software system. However, it does not address architectural models in a clear manner. The architectural description is a collection of products documenting a specific architecture. These products include different architectural models, and each model is related to views. The term software architecture is a widely used term, and for the purpose of this paper, software architecture is a set of architectural models. Such architectural models are a structural representation of a specific view. Each of these models is constructed using the methods established by their associated viewpoint. Examples of architectural models are: use case diagrams, class diagrams, and sequence diagrams.

.

3.2 Architectural model

The IEEE standard introduces a definition of architecture, this definition intended to encompass a variety of uses of the term architecture by recognizing their underlying common elements. This definition states that architecture is:

> "*The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution*"

Considering that a system may be an individual application, systems, subsystems, systems of systems, product lines, whole enterprises, and other aggregation of interest [1], we must distinguish different levels of abstraction in the software system. Let us express architectural model (AM_λ) having a certain level of abstraction, denoted as λ, which may be expressed as follows (note that this expression is similar to Perry and Wolf's model of software architecture [3]):

$$AM_\lambda = \{\text{Components, Relationships, Principles}\}$$

In a specific development of a software development, λ must be consistent among all the stakeholders of the system. AM_λ states that the basis of an architectural model is the design and evolution principles, but for some reason, in many occasions, these are not explicitly followed or considered. The principles for design and evolution are restrictions that prevail in the construction of architectural models. In addition, according to the conceptual model in the standard, these principles must be strictly bounded to the concerns of the stakeholder, which are closely related to quality attributes of the system. Although, architectural models consider components and relationships, and design and evolution principles, often this are not explicitly bounded (if any) to concerns. Then AM_λ can be extended, in order to show explicitly that the structure has principles of design and are bounded to some concerns of the stakeholder:

AM_λ = {Components, Relationships, Principles}
[stakeholder: Concern]

What is the impact of including concern on the components, relationships and principles of the architectural model? Obviously, an interest on an architectural model that fulfills specifications of the system is the target goal. An approach would be to assess the impact of design guidelines and principles. Components and the relationships between them are constrained by these principles along with viewpoint, views, which should be considered in the architectural model.

4. Enhancing the IEEE std 1471-2000

In this section, we introduce the enhancement of the conceptual model proposed in the standard. The effort is twofold. First, the model is enhanced through attributes in some classes; and a metrics class is added and associated with concerns, and the architectural model. Then we suggest a format to express the semantics of the enhance model using UML diagrams and OCL (Object Constrain Language). Second, a semi-formal notation for architectural models is introduced, under the assumption that an architectural model may participate in one or more views; and each view covers one or more concerns of a stakeholder. The main goal is to express explicitly, stakeholder, concern and view along with their role in specific architectural model.

4.1 Enhancing the conceptual model

Figure 1 shows the modified version of the conceptual model of architectural description introduced in [1]. Basically, the model was enhanced adding attributes in some classes. According to our architectural model expressed by AM_λ = {Components, Relationships, Principles} [stakeholder: Concern] that has guidelines and principles for its design and evolution which are related to concerns of a stakeholder. Therefore, "How can we assure that the design and evolution principles are considered in elaborating an architectural model?" A different rewording of the question may be if the architectural model is suitable enough for the system at hand. Suitability may be stated in terms of satisfying a set of criteria within these concerns. This means to some degree, an evaluation of the architectural model. There is a need to establish when an architectural model satisfies these criteria, and the principles for design and evolution.

In concrete terms, an architecture evaluation produces a report, the form and content of which varies according to the method used. Our approach proposes the consideration of metrics that allow decision-making and determining the degree of suitability of the architectural model.

Basically, the model was modified by adding a "Metrics" class associated with architectural model class, and a concern class. The metric class considers the criteria compliant with a specific concern. We do not suggest the use of specific metrics, the use of them depends on the selected criteria. For example in [8] and [9], the authors have developed metrics for specific criteria. It is important to note that a metric is an indicator of a particular feature within an architectural model, which addresses a specific concern. Now, we express our model using OCL format as shown in Table 1.

Table 1: Format for AD in OCL

context stakeholder **inv:**
self.name = stakeholder name
self.st_concern = stakeholder concern
self.st_desc = description
self.st_purpose = purpose of stakeholder
context concern
inv: self.has_associated ->notEmpty()
context viewpoint **inv:**
self.vpname = viewpoint name
self.vp_concern = viewpoint concern
self.vp_tech = associated modeling methods or analysis techniques
context view **inv:**
self.vwname = view name
self.vw_concern = view concern
self.vw_desc = view description
context architectural model **inv:**
self.amname = architectural model name
self.am_elements = components in the model
self.am_principles = principles for design
self.am_construles = rules of construction
self.has_associated ->notEmpty()

Summarizing, we have a framework and a format to express stakeholders, concerns, viewpoint, views, architectural models and metrics associated to an architectural model.

4.2 A semi-formal notation for architectural model description

Since the standard does not specify a way to express explicitly stakeholders, concerns, viewpoint and views; a semi-formal notation is proposed and applied to the architectural model introduced in section 3.1 (AM_λ = {Components, Relationships, Principles} [stakeholder: Concern]). This notation addresses the recommended practice provided in [1] and explained in Table 2

According to [1], the following definitions are considered:

- **Stakeholder**: An individual, team, or organization with interests in, or concerns relative to, a system.
- **Viewpoint**: A specification of the conventions for constructing and using a view.

- **View**: A representation of a whole system from the perspective of related set of concerns.
- **Concerns**: A concern expresses a specific interest in some topic pertaining to a particular system of interest.

In this semi-formal notation, an architectural model at some level λ, has a dependency on viewpoint, and a specific view. Viewpoints and concerns have a dependency on the stakeholder.

Table 2: Semi-Formal notation for AM

$AM_{\lambda}(Vp,v) = \{C, R, P\}$[stakeholder: Cn]
Where: AM_{λ}: architectural model at level λ Vp: viewpoint v: view C: components R: relationships between things P: principles guiding design and evolution am: architectural model Cn: concerns and Vp(stakeholder)

AM_{λ} is represented by components, relationships and principles. The latter are closely dependent on stakeholder concerns. The type of components and the relationships that conforms the architectural model are constrained by the level of abstraction, and by the viewpoint and view.

4.3 A brief discussion and example

Architectures serve as a communications vehicle in two ways. First, they are a common abstraction of the system providing a convenient -*lingua franca*- language that all stakeholders can speak and understand. Second, architectures serve as a communication vehicle by providing a technical "blueprint" for the system that is to be built, modified, or analyzed. Software architecture is a coherent and justified collection of system's earliest set of design decisions. These decisions will affect much of what the system will become [12], and must be taken into account in every architectural model.

The enhanced model and semi-formal notation proposed in this paper, expresses in a clear manner the aspects that are important for having a software architecture. We consider that stakeholders, concerns, viewpoint, and view must be expressed explicitly in the architectural models.

In order to demonstrate briefly our approach, we use a simple example of the early stages of development of an information system, where requirements are known through use cases, which were elaborated following a set of design principles and rules. The concern is correctness (extent to which a program fulfills its specification). The example follows OCL as basis. This example has a requirements engineer or analyst as stakeholder. This engineer/analyst has a specific concern, which can be expressed in the same way that we express viewpoint, view, and architectural model, and denoted as follows:

context correctness : concern **inv:**
    correctness.has_associated ->notEmpty()
context viewpoint **inv:**
    self.vpname = 'structural'
    self.vp_concern = 'correctness'
    self.vp_tech = 'UML v1.5'
context view **inv:**
    self.vwname = 'functional requirements '
    self.vw_concern = 'functionality of the system'
    self.vw_desc = 'describes the functional requirements and related non-functional requirements of system'
context architectural model **inv:**
    self.amname = 'Use Case'
    self.am_elements = 'actors, uses cases, associations'
    self.am_principles = 'Use Cases: Best Practice' -- Document UCBP
    self.am_construles = 'seven keys best practice' -- Document UCBP
    self.has_associated ->notEmpty()

In this document, Use Cases are considered as an architectural model. Use Cases are used in UML as a mean to define the requirements for software systems. In the section for architectural model (**context** architectural model **inv:**) the format states that Use Case elaboration should be performed following some principles and construction rules. The format adopts as principles and construction rules Use Cases: Best Practice (Document UCBP) [4] because it contains some practices addressing the correctness concern.

Another architectural model related to the functional view is the "Domain Model". According to Rational Unified Process® (RUP®), a domain model is a business object model that focuses on "product, deliverables, or events that are important to the business domain". The domain model typically shows the major business entities, their functional responsibilities, and the relationships among the entities [5]. In UML, a domain model is illustrated with a set of class diagrams without definition of the operations.

Consider we are working with the UP (Unified Process) methodology, and using the sample unified process artifacts and timing proposed by Larman [13]. Its is possible to express most of the information in a condensed form, as follows:

AM_{ls} (structural, functional)=
    ⟨Actors|UseCases, Associations, Document USBP⟩
    [analyst: correctness]

Then λ may be each of the four major phases: Ix: Inception, Ex: Elaboration, Cx: Construction and Tx: Transition. According with this methodology, subscript x may be s: start or r: refine. This example emphasizes several aspects. First, it states that the architectural model has a structural viewpoint and a functional view. Second, the architectural model has an analyst as a stakeholder, and his/her concern is correctness. Third, the architectural model under construction is a Use Cases model, and must follow Document UCBP as rules and principles for its design. Finally, λ = Is; which means that we are in inception stage, and starting the construction of the model.

It is worth highlighting that in both cases (using the format or semi-formal notation) we are prescribing an architectural model, not describing it.

5. Summary and future work

In this work, we have enhanced the conceptual model introduced in IEEE std 1471-2000, adding explicit attributes in each class, and associating a class metrics for architectural models related to concerns. In addition, we have proposed a semi-formal notation in order to express in a condensed way most of the information used in a specific architectural model.

The resulting approach highlights the ability to describe in a more explicit manner most of the important aspects that are involved in software architecture. Furthermore, our proposed model has the advantage to associate concerns and metrics, allowing evaluation of the architectural model.

Fundamentally, there are three reasons why software architecture is important: Communication among stakeholders, early design decisions, and transferable abstraction of a system [2]. Our approach helps to improve al least two of these points. First, the enhanced model, the format and the semi-formal notation offer a good way to express clearly, among stakeholders, the concepts stated in the IEEE standard. Second, the proposed model helps to express what design decisions must be taken into account when an architectural model is constructed, keeping in mind explicitly stakeholders and concerns. Finally, our framework provides a mean to define and associate metrics to concerns and to the architectural model.

Future work identifies the refinement of the proposed semi-formal notation and conceptual model. The work presented a conceptual model or framework, without addressing performance issues of a software architecture process. In this sense, work is underway in identifying, defining and establishing a software architecture process with the proposed framework as basis.

Acknowledgements

This research is supported by the National Council of Science and Technology of the State of Guanajuato (CONCYTEG) Project “Promoting Quality in the Software Industry: Human Resources, Research and Services” (GTO-2002-C01-5333)

L. F. Fernández Martínez is a professor at Universidad Autónoma de Ciudad Juárez, Electrical and Computer Department.

References

[1] Software Engineering Standards Committee of the IEEE Computer Society, *IEEE Recommended practice for architecture description of software-intensive systems*, IEEE Std 1471-2000, Approved 21 September 2000, IEEE-SA Standards Board, Print: ISBN 0-7381-2518-0 SH94869, PDF: ISBN 0-7381-2519-9 SS94869, available at (http://standards.ieee.org/).

[2] Bass, L., Clements, P., Kazman R., *Software Architecture in Practice*, SEI Series in Software Engineering, Addison Wesley, 5th printing May 1999.

[3] Perry, D. E. and Wolf, A. L., *Foundations for the Study of Software Architecture*, ACM SIGSOFT Software Engineering Notes, Vol. 17, No. 4, 1992, pp. 40-52

[4] Gottesdiener, Ellen, *Use Cases: Best Practice*, Rational Software June 2003, http://www.rational.com/products/whitepapers/474.jsp

[5] Menard, R., *Domain modeling: Leveraging the heart of RUP for straight through processing*, http://www.therationaledge.com/content/jun_03/t_domainmodeling_rm.jsp

[6] Chidamer, S., Kemerer, Ch., *A metrics suite for object oriented design*, IEEE Transactions on Software Engineering, Vol. 20, N0 6, June 1994, pp. 476-493

[7] Hyoseob, K., Boldyreff, C., *Developing software metrics applicable to UML models*, Centre for HCI Design, City University, Northhampton Square, London, ECIV 0HB, UK

[8] Garlan, D., Monroe, R. T., and Wile, D., *ACME: An Architecture Description Interchange Language*. Proceedings of CASCON '97 (1997), pp. 169-183.

[9] Medvidovic, N., and Taylor, R. N., *A Classification and Comparison Framework for Software Architecture Description Languages*. IEEE Transactions on Software Engineering, Vol. 26, No.1, January 2000, pp. 70-93.

[10] Clements, P., *A Survey of Architecture Description Languages*. 8th International Workshop on Software Specification and Design, Germany, March, 1996, pp. 16-25.

[11] Robbins, J. E., Medvidovic, N,. Redmiles, V., and Rosenblum, D. S., *Integrating Architecture Description Languages with a Standard Design Method*. In Proceedings of the 20th International Conference on Software Engineering (ICSE'98), 1998, pp. 209-218.

[12] Clements, P., Kazman, R., Klein, M., *Evaluating software architectures: Methods and case studies*, SEI Series in Software Engineering, Addison Wesley, 2000.

[13] Larman, G., *Applying UML and patterns: An introduction to object-oriented analysis and design and the Unified Process.* Prentice Hall, Second Edition, 2002.

[14] Bosch, J., *Design & Use of Software Architectures: Adopting and evolving a product-line approach*, Addison Wesley 2000