The words "structure", "organization", "relation" and "interaction" are used frequently. Although their precise meanings are different from each other, mostly all of them are used to mean the same thing. This way of using words implies that there is not yet a standard glossary for the literature of software architecture.

3. Classification of incorporating elements of definitions

To compare incorporating elements of definitions for software architecture from a general view point and free of any special methodology we engaged the present definitions in the system theory. The simplest system specification covers the set of elements and their relationships. A system can refer to anything. If we take the definition of software architecture as a system, the simplest decomposition for it comprises definition elements, features of each present element, relationships among defining elements and the collection of these elements. Based on this, every definition can be broken into three parts. According to the ongoing definitions for software architecture one might consider the followings in order to outline a comparison table:

1. Which elements are provided for the software architecture by the proposed definition? I.E. Which elements are covered in the software architecture of the respected definition
2. What kinds of details are mentioned for every element?
3. What kind of rationale is followed in the choice of elements?
4. What kind of relation governs the elements?
5. Finally, what configuration is provided for the software system by the whole chosen elements, relations etc.?

So any definition for software architecture consists of five sections:

1. The configuration of element sets and according relations in the definition
2. The elements of software architecture
3. The details of every element
4. The relationships among the elements of software architecture
5. The guidelines and logic behind the element choice.

Consequently, Table 1 is provided for the classification of all selected definitions.

First column of the presented table numbers the definitions and the remaining columns mapping the definitions into 5 aforementioned parts. A certain definition may not include all five parts. For instance, in the second definition "the details of elements" is not pointed out, so the respective cell for that definition is left blank.

In the next section, we pay attention to the comparison of the corresponding parameters in Table 1 and suggest a meta-model for each part based on the definitions and relationship comparisons and contrasts which shows how elements are interrelated inside any given group.

4. Comparison of the corresponding parameters of classification

In this section we want to compare the cells located in identical columns and hence to study the different definitions deeply. In the last column of Table 1 most of definitions use identical terms with the implication on the presence of; somehow, dominant principles on the system evolution which do not need any further examinations. The incorporated elements in similar columns are presented at table 2.

The concept and specification of each parameter should be developed prior to the comparison. Different definitions are suggested for any present entry in the table by miscellaneous groups.

For example, IEEE has proposed definitions for structure, component, and etc. But IEEE lonely or in joint with other groups has suggested a definition on the software architecture. We should search a higher standard for definitions regardless of any given literature. In order to achieve this, the definitions are compared in the pool of system theory. The engaged specifying tool is methodology independent. Next the definitions are translated into software systems and are compared one-by-one.

4.1. Relationship, Interaction, Connector, Interrelationship

For two given sets of A and B the Cartesian product is defined as: $A \times B = \{(X,Y): X \in A \wedge Y \in B\}$ The functional relationship of R from A to B is any possible subset of $A \times B$. ($R \subseteq A \times B$) If there is a relationship from A to B and from B to A as well then A is related to B or vice versa. The interrelationship between A and B is defined mathematically as: $A\ IR\ B \Leftrightarrow \exists R_1,\ A R_1 B \wedge \exists R_2,\ B R_2 A$. If in the relationship R of A to B something is being sent from A towards B the relationship is called a unilateral interaction or action while if something is mutually exchanged the relationship will be called an Interaction. For exchanging anything in computer field a path or connecting way between two elements is needed.

Connector in [5] is defined as an abstract machine which is used for Communication, Coordination and Cooperation among components.

Table 1. The proposed classification for incorporating elements in definitions.

Def NO.	Set of elements and Relations	Elements of Software Architecture	Details of each Element	Relation among their elements	Logic behind its choice
1	Structure	Software Elements	Externally Behaviours	Relationship	
2	Fundamental Organization	Components		relationships to each other and the environment	principles governing its design and evolution
3	set of significant decisions about the organization of	Structural and behavioural Elements	interfaces, behaviour	collaborations among those elements	the architectural style
4	A set of	processing, data, connecting elements			
5	structure	components		interrelationships	principles and guidelines to design and evolution
6	abstract system	functional components	behaviours and interfaces	interconnections	
7	collection	components	constraints	connections	satisfy the collection of system stakeholders' need
8	system or a collection of systems	software structures		interactions	desired set of qualities
9	high-level structure	architectural elements			satisfy the major functionality
10	basic framework, abstract system	functional components	Constraints	inter-relationship	rationale for choosing those components

Table 2. Incorporated elements in similar columns of classification.

Set of elements and relationships	Elements of Software Architecture	Details of elements	Relation between
Structure	Software Element	Properties	Relationship Interrelationship
Organization	Component	Interface	Interaction
Framework	Subsystem	Behaviour	Connector

So it is described as a port or run-time path for interactions among components in [3]. For mentioned definitions and descriptions a meta-model is developed by using an integrated modelling language and the class diagrams. For example, consider the former mathematical relationship. A and B are elements and relationship R from the element A toward the element B is established using an (X, Y) connector. It can be shown also by a class which bears the three specifications the source (A), the designation (B) and the connector (X, Y). As a result the developed model will be the same in Figure 1.

In the definitions of software architecture some terms of Relationship, Interrelationship, Interaction and Connector are used. One arisen question is that which of these terms fits best for the software architecture purpose? "Relationship" is a general word. The terms: Interaction, Coupling, Cohesion, Constraint, Function, Organization, Structure etc. are some kinds of relationships [8].

While the notion of action or interaction stepping beyond a simple Connector argues about the changeable objects too. Without any doubt, in software architecture context elements are represented as Black Boxes. The structure of exchanged objects in addition to the interaction among them should be specifiable and distinctive from other aforementioned interrelationships.

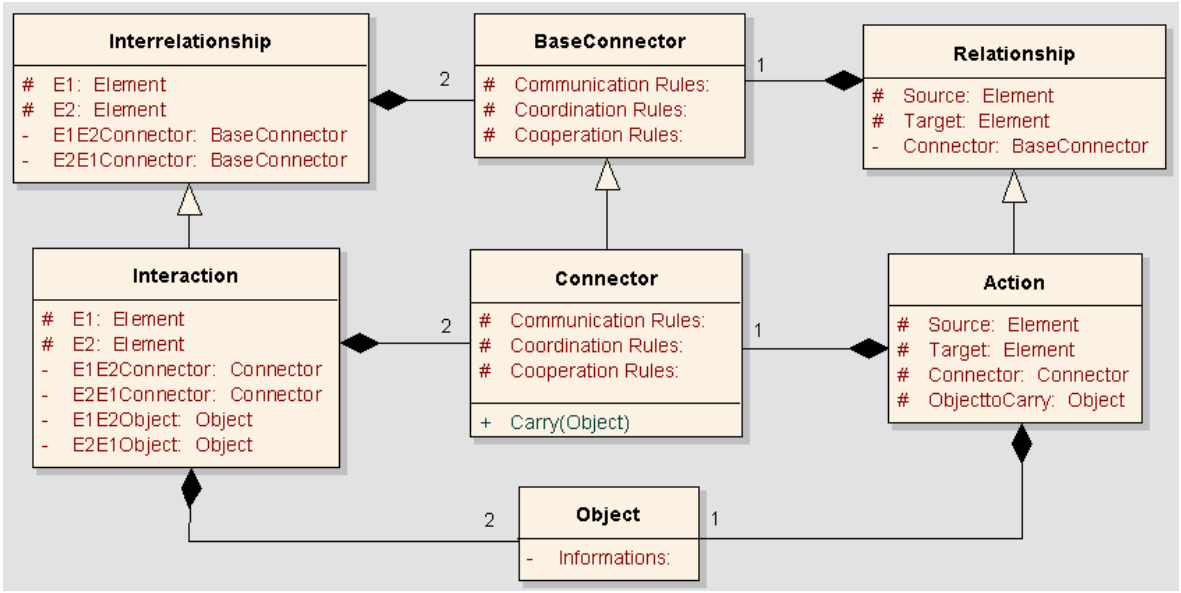

Figure 1: the suggested meta-model for Relation, Interrelationship, Interaction and Connector.

4.2. Elements, components and sub-system

The subject is that part of reality which is under study or research. Sub-subject is a part of subject. Factor is that part of the subject which exists independently and affects the subject significantly. A component is a sub-subject with at least two factors and one component [8]. The meta-model for the preceding definitions is as in figure 2.

In software systems, a software element can be a module, class, sub-system and etc. According to the definitions a component differs from a sub-system because a component is defined in the subject field while a sub-system is defined in the system field, implying that components emerge and survive independently from systems.

In [4] you can find more that 10 definitions or comparisons for "component". For example, in [12] a component is defined as follows: "A reusable part of software which is developed independently and combined with other components to construct bigger parts".

Each component first should incorporate a collection of interfaces to interact with other components and second the code should be executable [4]. Due to this definition many other elements such as compiled classes, library functions and etc. which incorporate connectors can be titled as "components".

This ignites the question: "which statement most suits for the definition of the software architecture?" The key point is that systems are developed according to their underlying components, but regarding the concept of a component in software systems we may not engage components in developing a software system and rather rely on alternative non-component based methods.

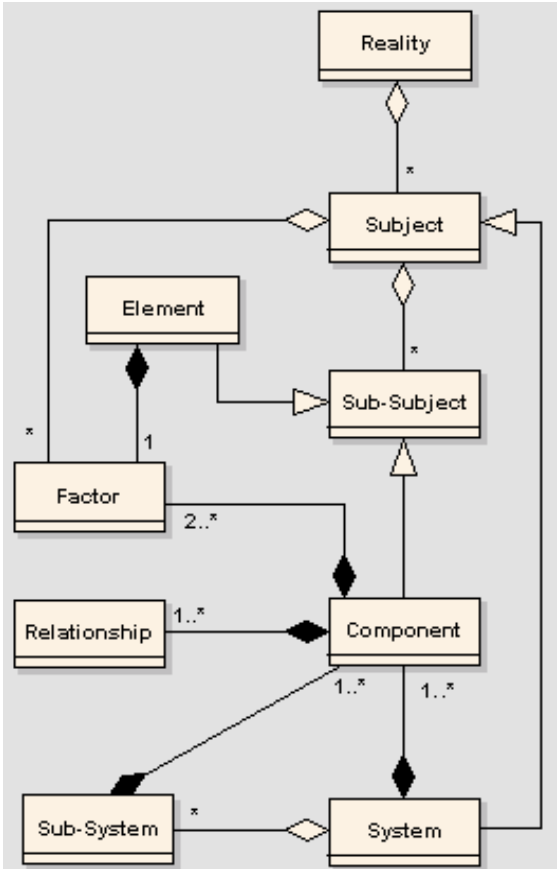

Figure 2. The developed meta-model for elements, components, the system and so on.

4.3. Property, Interface and Behaviour

The behaviour is a meaningful event originated from the subject or its components [8]. In [2], the behaviour is defined as the acting or reacting practices of an object in state change or message sending/receiving. In other words, it covers the

visible and testable actions of an object. Some points are implied in the definitions. First an object in the object oriented methodology can be anything. For further comparison and contrast between objects and components refer to [4]. Second, a component or an object cannot locate in an isolated context; that is; doubtlessly the objects and components carry behaviours. Third, behaviours are apparent and visible activities. As a result, intrinsic or extrinsic behaviours don't make any sense for viewers.

The property in [13] is defined as intrinsic or extrinsic property of an object. A property identifies the specifications of an object and differentiates if from other objects. An Attribute is an identified object. In most cases an attribute can replace a property and vice versa because they are interchangeable. However, in object oriented programming a property differs from an attribute. Properties are attributes for which Get and Set methods should be written, that is their change and accessibility are controlled by the programmer. For more information refer to [12].

In definitions, "property" is used for "object" while always "interface" is used for "component". "Interfaces" are the access points to the behaviours and properties of a component. In fact "interfaces" define the protocols and rules of access to behaviours and properties of a component through which other components can communicate with it. The interfaces are developed independent of the components' implementation which in turn allows us to change the implementation process without any need for altering the interfaces of a component. The detailed information is provided in [4].

Now the question, "which term or statement is more appropriate?" appears in the scenario. Clearly, for components, "interfaces" are the only visible parts and therefore the introduction of interfaces is sufficient. But for objects we should assign behaviours, extrinsic and visible properties which are attributes, properties and public methods.

For all mentioned definitions, similar to the earlier parts, a meta-model is suggested in figure 3.

4.4. Structure, Organizing and Framework

The simplest definition for "system" is a set of "components" and their relations which is shown as S= (E, R) where E is a non-empty set. Element is a general term and can refer to component, sub-system etc. As the number, diversification and inter relations of systems are expanding (the emergence of more complex systems) the collection of elements and system relations are arranged and represented logically based on a certain logic.

Level is a systemic threshold in which the components of a given subject are detailed in sub components and their inter relations. Levelling or hierarchy refers to the whole set of levels related to one subject. The "Structure" is the hierarchal representation of components and their inter relations.

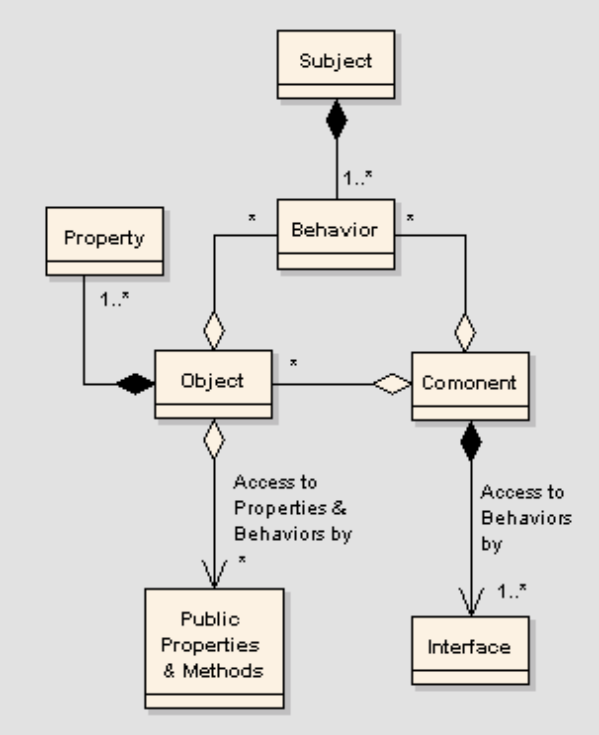

Figure 3. The suggested meta-model for behaviour, property, and interface.

If we develop a logic setting for the collection of system components and inter relations in a way that all existent system components and inter relations are located in this setting, the setting will be called a "Framework".

In fact, a framework is an implement for the set of system components and inter-relations and a basis for incorporating all systemic elements and inters relations. In software systems a framework is referred to a complete set of objects which are integrated to perform a special task jointly. For detailed information on software frameworks and their classifications refer to [7]. Some of best known frameworks are Mandaliov periodic table of natural elements, Zachman table of organizations. There exist some key points in the above-mentioned explanations and remarks. First, a framework can utilize several structures concurrently. Second a framework is a basis or house for all system elements and inter-relations with the implication of maximum generality for it. So a framework should engage as few as possible words, constraints and limitations in order to cover the maximum generality.

The question arisen here is that in the architecture besides suggesting a structure or framework for elements, what is done for organizing? I.E. are the allocation and coordination tasks among individuals and working groups incorporated in the software architecture or not?

5. Conclusion and future works

In this paper, some of architecture definitions and incorporating elements are analyzed and three corresponding meta-models are developed to facilitate the comparison of constituent elements inside them in order to provide better understandings from software architecture and the related elements and to die a cast in which more appropriate definitions and perceptions about the software architecture are materialized. This paper according to the testing nature of it over different definitions on software architecture and their incorporating elements can be used as a rigorous reference for defining software architecture and its elements.

Based on the suggested meta-models and the table for contrasting (or comparing) definitions , if one is going to develop a comprehensive and methodology-independent definition for the software architecture he/she can highlight more general parts in meta-models and then construct a general definition for the software architecture . For example, the choice of "element" can cover other identities like components and sub-systems. Further works can accentuate on endeavours of developing a new definition based on the ongoing methodologies in a way that it will allow the recognition and specification of any prospective architecture based on a given methodology.

In the ending part of any component comparing section some questions are raised for which finding answers may culminate at a unique framework for software architecture. It is obviously known that different conceptual schools and various practices have answered the questions indifferently. in spite of their common sense of the architecture their dictions are quite different.

10. References

[1] Bass, L., Clements, P., Kazman, R., *Software Architecture in Practice*, Second Edition, Addison-Wesley, 2003.
[2] Booch, G., Object-Oriented Analysis and Design with Applications, Second Edition, Addison-Wesley, 1994.
[3] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J., Documenting Software Architecture, Addison Wesley, 2002.
[4] Crnkovic, I., Larsson, M., Building Reliable Component-Based Software Systems, Artech House, 2002
[5] Fielding, R. T., Architectural Styles and the Design of Network-based Software Architectures, Department of Information and Computer Science, University of California, Irvine, 2000, from "http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation_2up.pdf"
[6] IEEE Standards Department, The Architecture Working Group of the Software Engineering Committee, Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, September 2000.
[7] Kaisler, S. H., Software Paradigms, John Wiley & Sons, 2005
[8] Klir, G., Facets of System Science, Plenum Press, 1991.
[9] McGovern, J., Ambler, S.W., Stevens, M.E., Linn, J., Sharan, V., Jo, E.K., A Practical Guide to Enterprise Architecture, Prentice Hall PTR, 2003
[10] Kruchten, P., the Rational Unified Process: An Introduction, Third Edition, Addison-Wesley, 2003.
[11] Software Engineering Institute (SEI), Carnegie Mellon University, How Do You Define Software Architecture, from "http://www.sei.cmu.edu/architecture/definitions.html".
[12] Souza, D. F. D., Wills, A. C., Objects, Components, Frameworks with UML: the Catalysis Approach, Addison-Wesley, 1999.
[13] Wikipedia Organization, Definition of Property in Computer Science, from "http://en.wikipedia.org/wiki/Property_(computer_science).