

# **A FORMAL CONCEPTUAL FRAMEWORK FOR DYNAMICS WITHIN CONTEXT OF COMPONENT-BASED SYSTEM FOR DESIGNING ROBUST SOFTWARE SYSTEMS & METRICS**

Meenakshi Sridhar and Naseeb Singh Gill

Department of Computer Science & Applications, M.D. University, Rohtak, India

## **ABSTRACT**

*It is a well-known fact that precise definitions play significant role in the development of correct and robust software. It has been recognized and emphasized that appropriately defined formal conceptual framework of the context/problem domain proves quite useful in ensuring precise definitions, including those for software metrics, which are consistent, unambiguous and language independent. In this paper, a formal conceptual framework for defining metrics for component-based system is proposed, where the framework formalises the behavioural aspects of the problem domain. The framework in respect of structural aspects has been discussed in another paper.*

## **KEYWORDS**

*Formal Conceptual Framework, Component, Assembly*

## **1. INTRODUCTION**

During the process of software development, the need for formalization—of all definitions in general, and of software metrics in particular, so as to overcome problems due to informal, imprecise or incomplete definitions—has been emphasized earlier, among others, by [2,3], [15], [1] & [16]. Further, these scholars, along with other scholars including [4], [13], [14], [20] and [22], have made significant contributions to the attempts in this direction.

In this paper, these contributions are extended by proposing a formal conceptual framework for defining metrics for a component-based system, where the framework takes into consideration the dynamics within problem domain. The framework for structural aspects has been discussed in [17].

[3] define a formalism based on set theory and graph theory for defining the coupling measures consistently, unambiguously and in language independent manner for object-oriented systems. [16] make similar attempt in respect of component-based software development using elementary mathematical concepts of set, relation, property and function. The approach followed in this paper is similar to that of [16]. The reason for following the approach as the basis of the work is that even by starting with a model based on assumptions of a perfect world (a deterministic, static etc) and of a perfect view (precise, complete, consistent, monotonic, factual, with agents having single shareable intelligence etc.) of it, one can gradually develop more realistic models and

robust solutions by incorporating imperfections (e.g. randomness) of problem domain and its knowledge (e.g. incompleteness/impresiseness). It can also be extended when the problem domain is dynamic and/or involves multi-agents. Another advantage of the approach is that it allows easy extension of the system when some atomic elements need to be considered later as composed also. For example, a professor of a university may be initially considered as atomic, with some specific attributes like subject specialization, date of birth, date of appointment as professor etc. However, the same professor, while being treated as a patient in the university clinic, may have to be considered in terms of organs of a human being, with each organ with its specific attributes. This type of extension of a system is facilitated by the approach.

The structure of the remainder of the paper is outlined next. The following section, which is the core of the paper, contains the details of the proposed formal conceptual framework. Its five subsections contain the details respectively of (i) meta model for set of system entities (ii) mathematical specification of functions between system entities (iii) mathematical specification of properties of functions between system entities (iv) specification of operators/ higher-order functions on set of functions and (v) mathematical specification of operations on general and binary relations between system entities. Finally, Section 3 concludes the presented work.

## 2. CONCEPTUAL FRAMEWORK

The conceptual framework by [17] discusses only the structure of the context, and is extended here to include formal conceptualisation of the behaviour/dynamics of constituents within the context.

As mentioned in [17], for developing software through component-based approach, Assembly, a slightly modified version of component, is taken as a unit instead of a component. An assembly is a software entity, which may not be a component ( i.e. it is not necessary that it conforms to the component model considered for the purpose), but, which may be obtained through composition which uses information only about provided and received interfaces(e.g. horizontal or partial vertical composition). The composition process initially starts only with some of the given components, and later uses recursively assemblies and components so obtained [4]. The advantage of using a set of assemblies instead of a set of components has been discussed in [19] and [17].

In order to represent in the meta model, the dynamics/behaviour of the problem domain, one need to also include some mathematical entities to represent various operations like composition of two components/ assemblies to get a new component/ assembly, or adding or deleting interfaces from a component or from a set of components to get the set of interfaces of a new component/ assembly. Similarly, one needs to represent the operations of adding /deleting parameters and parametric values of an interface to get new type of interfaces. The operations of addition/deletion in general may be represented by the mathematical operations of union, intersection and complementation. But, still in more general terms, any operation, whether it is composition or addition/deletion, is a function in mathematical sense. For example, if  $SC$  is the set of components, then composition, in mathematical sense, is a function, say,  $Comp: SC \times SC \rightarrow SC$ . Similarly, the operation of adding some attributes to the set of attributes of a given interface may be represented as a function Add-Attribute-Interface:  $SI \times P(SA) \times P(SA) \rightarrow SI \times P(SA)$ , where,  $SI$  represents the set of interfaces, and  $P(SA)$  represents set of sets of attributes. An element of the first occurrence  $P(SA)$  on L.H.S. may represent the set of already associated attributes of an interface and an element of the second occurrence of  $P(SA)$  on L.H.S. may represent the set of attributes to be added, and an element of the  $P(SA)$  on R.H.S. may represent the new set of attributes obtained after addition.

Also, the functions may have properties, e.g. a function may be 1-1 or onto. Further, a function as an operation on a set may have property of, say, commutativity.

For the definition of the meta-model, it is assumed that a software system  $S$  is required to be developed to provide all the required services, say  $Serv$ . Further, for this purpose, it is also assumed that initially a set of components, say  $CR$ , is given. And, on the basis of judgement, a subset  $Comp(S)$  of  $CR$  is selected to develop the required software  $S$ .

As a first step for formalization, an identifier need to be associated with each of the entities, property of an entity, relation between any two entities and property of a relation etc., and also, to each of the sets, which may be required in the discussion, of these entities, of relations and of properties. Let  $S_i, i=1$  to  $n$ , be the identifiers associated with required services, and let  $C_j, j= 1$  to  $m$ , be the identifiers associated with the initially provided components. Then, formally,

$$Serv = \{S_1, S_2, \dots, S_i, \dots, S_n\} \text{ and } CR = \{C_1, C_2, \dots, C_j, \dots, C_m\}$$

In view of the fact that the proposed meta model is to include formal conceptual framework of the behaviour/ dynamics of the problem domain through various operations on the underlying set  $CR$ , which then evolves into larger sets, and finally into the required software  $S$ ; let  $SA$  denote the set of assemblies, that represents the evolving (dynamically changing) set, the initial value of which is  $CR$ . However, a member of  $SA$  may be restricted to be a component only, where it is either an initially given component or is obtained by composing two members of  $SA$ . Also, let  $Assemb(S)$ , the initial value of which is taken as  $Comp(S)$  denote the set of selected assemblies, at a particular point of time, to construct the required software  $S$ . Each of the sets to be defined below evolves as  $SA$  evolves.

The meta-model for the Structure of Set of System Entities proposed by [17] consists of 4-tuple  $(E, Prop(E), Rel(E), Rel-prop(E))$ , in which (a)  $E$  represents the set of system entities (in this case, an entity  $a_i$  is an assembly);(b)  $Prop(E)$  denotes the properties of the elements from  $E$ ; (c)  $Rel(E)$ denotes the set of relations between the entities constructed out of elements of  $E$ ; and (d)  $Rel-prop(E)$  denotes the set of properties of the relations between the entities constructed out of elements of  $E$ .

In view of the discussion in the preceding paragraph, the proposed meta model also includes (e) the set of functions, which mathematically may represent various operations like composition of members of components or assemblies. A function is defined by (i) its domain, say  $D$  (ii) range, say,  $R$  and (iii) the rule that associates to each member of  $D$  a unique member of  $R$ . Also, the meta-model includes (f) the set of properties of functions mentioned under (e).

The process of formal definition of the proposed meta-model for the dynamics/behaviour of the context starts with the following semi-formal (partly expressed in natural language) definition.

## 2.1. A Meta Model for Set of System Entities (SSE)

**Definition 1: (A meta-model for Set of System Entities)** The 6-tuple

$$\{E, Prop(E), Rel(E), Rel - Prop(E), Func(E), Func - prop(E)\}$$

is called a meta-model for Set of System Entities (SSE), where

- a.  $E$  represents the set of system entities (in this case, an entity  $a_i$  is an assembly);
- b.  $Prop(E)$  denotes the properties of the elements from  $E$ ;
- c.  $Rel(E)$ denotes the set of relations between the entities constructed out of elements of  $E$
- d.  $Rel-prop(E)$  denotes the set of properties of relations on  $E$

- e. Func (E) denotes the set of functions, with domain and range for each being constructed out of elements of E
- f. Func-prop (E) denotes the set of properties of functions in Func(E).

Out of the six elements/ components of meta-model included in the Definition 1 above, the first four components, which are about the structure of the context, have been described mathematically in [17].

In the ensuing sections, the semi-formal definitions given above of Func (E) and Func-prop (E) about the dynamics/ behaviour within the context are formalized in terms of mathematical concepts of set and relation.

In [17], the set of system entities E is formally defined by the following equation:

$$E = SA \cup Serv \cup I(SA) \cup I(Serv) \cup Param (SA) \cup Param (Serv).$$

where I(SA) denotes the set of all interfaces of SA, I(Serv) the set of all interfaces of Serv, Param (SA) the set of all parameters of SA, and Param (Serv) the set of all parameters of Serv.

Also, two other sets viz. Types (E) and Val (E) are associated with E, denoting respectively types of elements of E and values which may be associated with a property of an assembly, interface or a parameter in E. It may be noted that the set Val (E) can only be decided and computed during the actual process of developing software using component-based approach (CBSE).

For formal specification of Func (E) and Func-prop (E), the following notations are required. The notations (mathematical specifications) are given only in terms of a generic type T. This is being done in view of the fact that mathematical specifications of different types are similar. T may denote, at one time, only one of different types. In this connection, it may also be noted that for all the elements of a particular type of entities, the properties are same; whereas, for two elements of different types, properties may be different. Let EnT denote the set of entities of type T and PEnT denote the ordered set of properties of elements of EnT with PEnT = (PT1, PT2, ..., PTi, ..., PTtm), where tm is the number of properties of any entity in EnT. Further, V-PEnT(i) denote the set of values of Property PTi, for i= 1, 2, ..., tm. Then, V-PEnT, the set of values associated with all properties of an entity of Type T, is given by the equation  $V-PEnT = \cup V-PEnT(i)$ , union being taken over all properties PTi, of an entity T. For the purpose of defining Val(E), the set of all possible values of properties of elements in E, let V-PEnT, instead be denoted as V-PEn (T), in which T is taken explicitly as parameter. Then  $Val(E) = \cup V-PEn (T)$ , where union is taken over all types T of elements of E.

At any point of time, to each entity eTj and property PTi, a unique value vik is associated. This fact may be mathematically specified as: value (eTj, PTi) = vik. It may be further expressed as the function

$$Fun-Pro-Val-T-i : EnT \rightarrow V-PEnT(i), \text{ for property } PTi, \text{ for all } i\text{'s.} \quad \dots (A)$$

This may be further generalized to function

$$Fun-Pro-Val-T: EnT \rightarrow V-PEnT(1) \times V-PEnT(2) \times \dots \times V-PEnT(m) = (let) V-PEnT \dots (B).$$

In the above, the type name T is used as suffix of various names such as EnT, PEnT, V-PEnT etc. However, T is to be used as parameter in the further discussion, therefore, instead of EnT, PEnT, V-PEnT(i), V-PEnT, and Fun-Pro-Val-T, respectively the notations En(T), PEn(T), V-PEn(T, i),

V-PEn(T), and Fun-Pro-Val(T) are used. The specification of functions between system entities of E, is discussed in detail.

## 2.2. Mathematical Specification of Functions between system entities.

The description of specification of functions given below is merely an illustration and that it is not exact and complete. Rather, it cannot be exact and complete in view of the facts that all the functions cannot be determined in advance as these heavily depend on the specific requirement and usage context. Therefore, the exact and complete specification of functions is determined by the team of designers and potential users of the system as per requirements and context.

Also, though there may be a number of types of functions that can be considered for mathematical specification, yet the functions are restricted to having their domains and ranges as only the subsets of the set:

$$E \cup \text{Val}(E) = (SA \cup \text{Serv} \cup I(SA) \cup I(\text{Serv}) \cup \text{Param}(SA) \cup \text{Param}(\text{Serv})) \cup \text{Val}(E).$$

For the purpose of the rest of the discussion, the following assumptions are made:

1. The following elementary operations on sets are in-built in the system: union, intersection, cross-product of two or more given sets, complement of a set in some given set, and set-constructor which builds a set out of given elements etc.
2. The following elementary relations on sets are in-built in the system: Equality of two sets, is-a-subset-of, is-a-proper-subset-of etc.
3. For all natural numbers n and i, with  $1 \leq i \leq n$ , the following projection functions are built-in:

**Func-Proj-n-i:  $A_1 \times A_2 \times \dots \times A_i \times \dots \times A_n$  (  $A_i$ ),** such that

$$\text{Func-Proj-n-i}((a_1, a_2, \dots, a_i, \dots, a_n)) = a_i.$$

The function **Func-Proj-n-i** maps an n-tuple from  **$A_1 \times A_2 \times \dots \times A_i \times \dots \times A_n$**  to its ith component, a member of  $A_i$ .

4. A function  $g: X \rightarrow Y$  can be defined/ constructed when either all the pairs  $(x, g(x))$  or some rules for forming such pairs are given.
5. Through in-built mechanisms, the composition function fog of two given functions  $f: Y \rightarrow Z$  and  $g: X \rightarrow Y$  can be defined such that  $(fog)(x) = f(g(x))$ .  
Through in-built mechanisms, the composition function fog of two given functions  $f: Y \rightarrow Z$  and  $g: X \rightarrow Y$  can be defined such that  $(fog)(x) = f(g(x))$ .
6. The elements in the set  $E \cup \text{Val}(E)$  are distinctly represented in such a manner that the type of an element is part of its representation and the type of an element can be extracted easily from the representation. One of the ways in which the objective can be achieved is by representing each element of  $E \cup \text{Val}(E)$  as an ordered pair  $(e, t)$ , where e represents the name of the element and t represents the name of the type of e. Then using projection functions defined earlier, the type can be specified and determined. In general, it is assumed that for a set S involving elements of different types with the types being members of **Types =  $\{T_1, T_2, \dots, T_k, \dots, T_n\}$** , it is possible to determine the type of a given element of S.

Next, the classes of (more complex) functions that are specified under the above-mentioned assumptions are enumerated:

1. *Func-Type* :  $SA \cup I(SA) \text{ Param}(SA) \cup \text{Val}(E) \rightarrow \text{Types}$  , where each type being a member of the set  $\text{Types} = \{T_1, T_2, \dots, T_i, \dots, T_n\}$ . The function *Func-Type*:  $S \rightarrow \text{Types}$  returns the type of an element of S. The function *Func-Type* may be specified in terms of projection functions *Func-Proj-n-i* for appropriate values of n and i.
2. *Func-Type-Properties*:  $\text{Types} \rightarrow \cup \text{PEn}(T)$ , where union is taken over  $T \in \text{Types}$ , returns the set of properties associated with each type from  $\text{Types}$ . One of the ways in which this type of functions can be constructed and can be specified is: Consider each element of  $\text{Types}$  as consisting of ordered pairs (T, Set-of-Properties-of-T), where T is type-name and Set-of-Properties-of-T is an ordered n-tuple of properties of an element of the type with type-name T, and n is the (assumed) number of elements in Set-of-Properties-of-T.
3. *Func-Element-Properties*:  $E \rightarrow \cup \text{PEn}(T)$ , where union is taken over  $T \in \text{Types}$ , returns the set of properties associated with an element of E. Actually, the function can be specified through the equation: *Func-Element-Properties* = *Func-Type-Properties*  $\circ$  *Func-Type*
4. (i) *Func-Interface*:  $SA \rightarrow I(SA)$  such that *Func-Interface* (a) =  $I_a$ , the set of interfaces of a for  $a \in SA$  (ii) *Func-Provided*:  $SA \rightarrow PI(SA)$  such that *Func-Provided* (a)= $PI_a$ , for  $a \in SA$  and (iii) *Func-Required*:  $SA \rightarrow RI(SA)$  such that *Func-Required* (a)= $RI_a$  for  $a \in SA$ . The mechanism for specifying *Func-Interface* etc. is similar to the mechanism used earlier for specifying *Func-Type-Properties*. For this purpose, each element of SA may be represented as a 3-tuple ( a,  $PI_a$ ,  $RI_a$ ). Then using appropriate projection function, each of the two functions *Func-Provided* and *Func-Required* can be specified. And the function *Func-Interface* can be specified by taking functional value  $\text{Func-Interface}(a) = \text{Func-Provided}(a) \cup \text{Func-Required}(a)$ , for each  $a \in SA$ .
5. *Func-Compose-Assembly*:  $\text{Rules} \times SA \times SA \rightarrow SA$ , where  $\text{Rules} = \{\text{Horizontal-binding, Vertical-binding, Partial-Vertical-binding, } \dots\}$ , returns an assembly obtained after composing two given assemblies according to some given rule in Rules [4].

One of the possible rules for composing and specifying assemblies may be described as follows. Let C be an assembly obtained by composing two assemblies, say, A and B. The interfaces  $PI_c$  and  $RI_c$  for component C are obtained from the provided and required interfaces of A and B as follows: First, find those required interfaces of A which are provided interfaces of B, and also find those required interfaces of B which are provided interfaces of A i.e. find  $RI_a \cap PI_b$  and  $RI_b \cap PI_a$ . Then, one may take  $PI_c = PI_a \cup PI_b$  and  $RI_c = (RI_a \cup RI_b) \sim ((RI_a \cap PI_b) \cup (RI_b \cap PI_a))$ .

The interfaces of the assembly obtained through the composition may be added and/or deleted through set operations of union, intersection and complementation. Thus, the composition of assemblies/components described above is formally specified by sets and set operations.

6. *Fun-Weight-Assembly*  $SA \rightarrow \text{Real}$  (may be restricted to [0, 1]). One of the ways in which one may define *Fun-Weight-Assembly* (a) for  $a \in SA$  is through associating (relative, but non-negative numbers as) weights with interfaces in  $PI \cup RI$ . Let  $w_i$  be the weight associated with interface i. Then

$$Fun-Weight-Assembly(a) = \sum^P W_{p_i} - \sum^R W_{r_i}$$

where  $w_{p_i}$  is the weight associated with an interface  $p_i \in PI_a$  and  $w_{r_i}$  is the weight associated with an interface  $r_i \in RI_a$ ;  $\sum^P$  denotes summation over all interfaces in  $PI_a$ , and  $\sum^R$  denotes summation over all interfaces in  $RI_a$ .

It may be noted that the above definition of Fun-Weight-Assembly is quite simplistic and is only given here just to explain the essential idea of how an assembly may be evaluated and specified. Other factors that may be taken into consideration for the function include the effort required to convert an assembly into a component according to the component model under consideration.

In respect of finding values for a given entity/attribute, the following two types of functions are already defined:

7. **Fun-Pro-Val-T-i : EnT ( V-PEnT(i) )** , which maps an entity of EnT to its value for its ith property PTi,
8. **Fun-Pro-Val-T: EnT ( V-PEnT(1) × ... × PTi × ... × V-PEnT(m) )** , which maps an entity of EnT to ordered set of its values for properties PT1, PT2, ....., PTi, ....., PTm. More explicitly, if eT is an element of type T, then **Fun-Pro-Val-T (eT) = (v1T, v2T, ....., viT, ....., vtmT)** , where viT is some value of property PiT.

In order to define functions for values of an element of E, let type T be explicitly indicated, in the above type of functions, so that this type of functions can be rewritten as

$$Fun-Pro-Val(T): En(T) \rightarrow V-PEnT(1) \times V-PEnT(2) \times \dots \times PTi \times \dots \times V-PEnT(m)$$

If Types = {T1, T2, ....., Tk, ....., Tn} is the set of all types, these functions may be restricted to the functions:

$$Fun-Pro-Val: E ( \cup (V-PEnT(1) \times \dots \times PTi \times \dots \times V-PEnT(m)) ) ,$$

where union is taken over elements of Type. Then, in order to determine value of an element e of E, first its type, say Tk, is determined. Then **Fun-Pro-Val(e) = Fun-Pro-Val-Tk(e)** .

### 2.3. Mathematical Specification of properties of functions between system entities.

- Author **Function-specification-methods = {Table/Graph, formula, implicit, algorithm, lambda-calculus, .....** }
- **Function-types =** {constant, identity, linear, polynomial, rational, algebraic, analytical, smooth, continuous, measurable, injective, surjective, bijective/invertible}
- Operation-arities = { unary, binary, ternary, quaternary }  $\cup$  {n-ary, with n being an integer  $\geq 5$ }
- **Binary-operation-properties = {commutative/ non-commutative, associative/ non-associative}**

A function will be specified as a tuple:  
**(name, domain, codomain, rule/ method, type, arity, ...)**

It is assumed that function names are assigned uniquely in the sense the operations 'sum of two numbers' and 'sum of two vectors' will be given different names e.g. respectively sum-num and sum-vector. Thus, names are assigned in such a way that attribute 'name' may serve as a key.

Some component of the tuple may have a special value say NULL if the property is not relevant. For example, property-of-binary-operation is null, if either the function is not an operation on some set or if the value of Operation-arity  $\neq$  binary. Also, if corresponding to a property-value there are further possible properties, then property will be represented as a tuple. For example, the value 'binary' of operation-arity will be represented as (binary, {commutative/ non-commutative, associative/ non-associative}). Further, value of a tuple may be a pointer pointing to a procedure, if the value is a method/ procedure.

Using the above facts, a program may be written as specification of the method for finding properties and their values.

## 2.4. Specification of Operators/ Higher-order functions on Set of Functions

The Using the specification for a function as given above and in-built projection operations, one can define the following operators/ Higher-order functions, the arguments of each of which is a function:

1. Domain-of-function
2. Co-domain-of-function
3. Method-of-function
4. Type-of-function
5. Arity-of-operation
6. Property-of-binary-operation
7. Composibility-of-a-function1-with-function2, i.e. Is (function1  $\circ$  function2) defined?
8. Composition-of-functions.

As an example, Composibility-of-a-function1-with-function2 can be specified through the program segment:

If domain (function1)  $\subseteq$  range (function2) then True

## 2.5. Mathematical Specification of operations on general and binary relations between system entities.

$X_1, X_2, \dots, X_i, \dots, X_k$ , i.e.  $R \subseteq X_1 \times X_2 \times \dots \times X_i \times \dots \times X_k$ .

**Set-of-operations-on-k-ary-relations = {union, intersection, complementation, inverse}** ,

where all considered **k-ary** relations are subsets the product of sets  $X_1, X_2, \dots, X_i, \dots, X_k$ , of the same  $k$  sets  $X_i$ , in which even the order of the sets also remains the same. When  $k=2$ , the set may be called 'Set-of-operations-on-binary-relations'. Further, if  $K=2$  and  $X_1 = X_2$ , additionally, one can define the following set:

Set-of-operations-on-binary-relations-in-a-set = { Composition, Reflexive-closure, Reflexive-reduction, Transitive-closure, Transitive-reduction, Reflexive-transitive-closure, Reflexive-transitive-symmetric-closure}.

The operations included in Set-of-operations-on-binary-relations, may be defined as follows.



If  $R$  and  $S$  are two binary relations from  $X$  to  $Y$ , then each of the following is a binary relation from  $X$  to  $Y$ :

- Union:  $R \cup S \subseteq X \times Y$ , defined as  $R \cup S = \{(x, y) \mid (x, y) \in R \text{ or } (x, y) \in S\}$ . For example,  $\geq$  is the union of  $>$  and  $=$ .
- Intersection:  $R \cap S \subseteq X \times Y$ , defined as  $R \cap S = \{(x, y) \mid (x, y) \in R \text{ and } (x, y) \in S\}$ .
- Complementation: If  $R \subseteq X \times Y$ , then the complement of  $R$ , denoted as Complement ( $R$ ) in  $X \times Y$  is defined as  $x$  Complement( $R$ )  $y$  if not  $x R y$ . For example, on real numbers,  $\leq$  is the complement of  $>$ .
- But Inverse or converse:  $R^{-1}$ , defined as  $R^{-1} = \{(y, x) \mid (x, y) \in R\}$  is a binary relation from  $Y$  to  $X$ . For example, "is less than" ( $<$ ) is the inverse of "is greater than" ( $>$ )

The above-mentioned four operations can be generalized when  $R$  and  $S$  etc. are  $k$ -ary relations, for  $k \geq 3$ , instead of being just binary, as discussed below:

Let  $R$  and  $S \subseteq X_1 \times X_2 \times \dots \times X_i \times \dots \times X_k$  be  $k$ -ary relations, then the operations on relations, viz. union  $R \cup S$ , Intersection  $R \cap S$ , complementation etc. are defined as follows:

- Union:  $R \cup S \subseteq X \times Y$ , defined as  $R \cup S = \{(x_1, x_2, \dots, x_i, \dots, x_k) \mid (x_1, x_2, \dots, x_i, \dots, x_k) \in R \text{ or } (x_1, x_2, \dots, x_i, \dots, x_k) \in S\}$  is a the  $k$ -ary relation. For example,  $\geq$  is the union of  $>$  and  $=$ .
- Intersection:  $R \cap S \subseteq X \times Y$ , defined as  $R \cap S = \{(x_1, x_2, \dots, x_i, \dots, x_k) \mid (x_1, x_2, \dots, x_i, \dots, x_k) \in R \text{ and } (x_1, x_2, \dots, x_i, \dots, x_k) \in S\}$  is a the  $k$ -ary relation.
- Complementation: If  $R \subseteq X_1 \times X_2 \times \dots \times X_i \times \dots \times X_k$ , then the complement of  $R$ , denoted as Complement ( $R$ ) in  $X_1 \times X_2 \times \dots \times X_i \times \dots \times X_k$  is defined as  $(x_1, x_2, \dots, x_i, \dots, x_k) \in$  Complement ( $R$ ) if not  $(x_1, x_2, \dots, x_i, \dots, x_k) \in R$ .
- Inverse or converse:  $R^{-1}$ , defined as  $R^{-1} = \{(x_k, x_{k-1}, \dots, x_i, \dots, x_1) \mid (x_1, x_2, \dots, x_i, \dots, x_k) \in R\}$  is an  $k$ -ary relation, i.e.  $R^{-1} \subseteq X_k \times X_{k-1} \times \dots \times X_i \times \dots \times X_1$

Next, we discuss specification of operations on relations in a set  $X$ , i.e. when  $R \subseteq X \times X$ :

- Composition:  $S \circ R$ , also denoted  $R; S$  (or more ambiguously  $R \circ S$ ), defined as  $S \circ R = \{(x, z) \mid \text{there exists } y \in Y, \text{ such that } (x, y) \in R \text{ and } (y, z) \in S\}$ . The order of  $R$  and  $S$  in the notation  $S \circ R$ , used here agrees with the standard notational order for composition of functions. For example, the composition "is mother of"  $\circ$  "is father of" yields "is paternal grand-mother of", while the composition "is father of"  $\circ$  "is mother of" yields "is maternal grandfather of".
- Reflexive-closure:  $R^=$ , defined as  $R^= = \{(x, x) \mid x \in X\} \cup R$  or the smallest reflexive relation over  $X$  containing  $R$ . This can be proven to be equal to the intersection of all reflexive relations containing  $R$ .
- Reflexive-reduction:  $R^\neq$ , defined as  $R^\neq = R \setminus \{(x, x) \mid x \in X\}$  or the largest irreflexive relation over  $X$  contained in  $R$ .
- Transitive-closure:  $R^+$ , defined as the smallest transitive relation over  $X$  containing  $R$ . This can be seen to be equal to the intersection of all transitive relations containing  $R$ .
- Transitive-reduction:  $R^-$ , defined as a minimal relation having the same transitive closure as  $R$ .
- Reflexive transitive closure:  $R^*$ , defined as  $R^* = (R^+)^=$ .
- Reflexive-transitive-symmetric-closure:  $R^=$ , defined as the smallest equivalence relation over  $X$  containing  $R$ .

### 3. CONCLUSIONS

In this paper, the concerns, earlier expressed by some researchers are further emphasized, in respect of the desirability of defining some formal conceptual framework of the context in order to have unambiguous, precise and language independent definitions, including those of system metrics, which in turn help in designing robust software. A formal conceptual framework of the dynamics within the problem domain is defined; for which instead of component, an assembly—a slightly modified and more general concept—is taken as basic building block for design and development of software. The advantage of the proposed framework is that it is more efficient than a framework in which ‘component’ is necessarily taken as building block. Another advantage of proposed framework is its scalability. The definition of the proposed framework is given under the assumption—common for most of the software development endeavours so far—that the problem domain and its knowledge, both are perfect. However, for developing realistic models and robust solutions, it is essential to take into consideration both imperfections of problem domains (e.g. inherent randomness of domain) and of its knowledge (e.g. incompleteness/impresiseness). The proposed framework can be easily extended to the cases when the problem domain is imperfect and/or knowledge of the problem domain is imperfect, thereby providing solid foundations for developing robust software.

### ACKNOWLEDGEMENTS

The authors are thankful to Prof. Manohar Lal, SOCIS, IGNOU, New Delhi (India) for his valuable suggestions.

### REFERENCES

- [1] A. Baroni, S. Braz and F. Abreu, Using OCL to formalize object-oriented design metrics definitions. Proceedings of ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, 2003.
- [2] L. Briand, S. Morasca and V. Basili, Property-based Software Engineering Measurement, IEEE Transactions On Software Engineering, Vol. 22, No. 1, January 1996.
- [3] L. Briand, J. Daly and J. Wust, A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Transactions on Software Engineering, 25(1), 91-121, 1999.
- [4] Crnkovic, S. Sentilles, A. Vulgarakis, and M. Chaudron, A Classification Framework for Software Component Models, IEEE Transactions On Software Engineering, Vol. 37, No. 5, September/October 2011.
- [5] N. Gill, Importance of Software Component Characterization For Better Software Reusability, ACM SIGSOFT Software Engineering Notes Vol. 31 No. 1, January 2006.
- [6] N. Gill, and P. Grover, Component-Based Measurement: Few Useful Guidelines, ACM SIGSOFT Software engineering Notes, Vol. 28 Issue 6. November 2003.
- [7] N. Gill, and P. Grover, Few Important Considerations for Deriving Interface Complexity Metric for Component-Based Software, Software Engineering Notes, Vol. 29, 2004.
- [8] M. Goulão, F. Abreu, Formalizing metrics for COTS, Department of Informatics, Faculty of Sciences and Technology, New University of Lisbon, 2825-114 Monte de Caparica, Portugal, 2005a.
- [9] M. Goulão, and F. Abreu, Formal Definition of Metrics upon the CORBA Component Model. First International Conference on the Quality of Software Architectures (QoSA) 2005b.
- [10] G. D. Jenson, J. Dietrich, H. Guesgen, A Formal Framework to Optimise Component Dependency Resolution, 2010 Asia Pacific Software Engineering Conference, 2010.
- [11] Z. Liu, and H. Jifeng, Mathematical Frameworks for Component Software, World Scientific, 2006.
- [12] C. Mair, and M. Shepperd, Human Judgement and Software Metrics: Vision for the Future ICSE '11, May 21–28, 2011, Waikiki, Honolulu, HI, USA, 2011.

- [13] W. McUumber and B. Cheng, A General Framework for Formalizing UML with Formal Languages, Proc. of IEEE International Conference on Software Engineering (ICSE01), May 2001, Toronto.
- [14] M. Pereplechikov, C. Ryan, K. Frampton, and H. W. Schmidt, A Formal Model of Service-Oriented Design Structure, Proceedings of the Australian Software Engineering Conference (ASWEC'07), 2007.
- [15] R. Reißing, Towards a model for object-oriented design measurement. Proceedings of ECOOP Workshop on Quantitative Approaches in Object Oriented Software Engineering, 2001.
- [16] C. Serban, A. Vescan and H. Pop, A Conceptual Framework for Component-based System Metric Definition, 9th RoEduNet IEEE International Conference 2010.
- [17] M. Sridhar, and N. Gill, A Formal Conceptual Framework for Structure of Context of Component-Based System for Designing Robust Software Systems & Metrics (communicated 2015).
- [18] Y. Tu, D. Li, F. Li and S. Zheng, A Formal Framework for Component-Based Embedded System, 2010 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Montréal, Canada, July 6-9, 2010.
- [19] K. Wallnau, and J. A. Stafford, Dispelling the Myth of Component Evaluation, in Building Reliable Component-Based Software Systems, Edited by I. Crnkovic and Larsson pp. 157-177, 2002. Boston, London: Artech House, pp. 157-177, 2002.
- [20] J. Woodcock, P. Larsen, J. Bicarregui and J. Fitzgerald, Formal Methods: Practice and Experience, ACM Computing Surveys, Vol. 41, No. 4, Article 19, Publication date: October 2009.
- [21] J. Wust, L. Briand, and J. Daly, A Unified Framework for Cohesion Measurement in Object-Oriented Systems. Empirical Software Engineering: An International Journal, 3(2), 65-117, 1998.
- [22] J. Zhixiong, L. Qian, and X. Pen, A Formal framework for description of semantic web services , Seventh International Conference on Computer and Information Technology 2007 IEEE, DOI 10.1109/CIT.2007.24.